Nuitka Release 0.4.1 (Draft)

This release is the first follow-up with a focus on optimization. The major highlight is progress towards SSA form in the node tree.

Also a lot of cleanups have been performed, for both the tree building, which is now considered mostly finished, and will be only reviewed. And for the optimization part there have been large amounts of changes.

New Features

- Support for FreeBSD.

  Actually only the minor issue of default compiler detection was standing in the way of this, and more generally "Unix" should work now.

- Python 3.3 experimental support

  - Now compiles many basic tests. Ported the dictionary quick access and update code to a more generic and useful interface.
  - Added support for `__qualname__` to classes.
  - Small compatibility changes. Some exceptions changed, absolute imports are now default, etc.
  - For comparison tests, the hash randomization is disabled.

- Python 3.2 support has been expanded.

  The Python 3.2 on Ubuntu is not providing a helper function that was used by Nuitka, replaced it with out own code.

Bug fixes

- Default values were not "is" identical.

```
def defaultKeepsIdentity( arg = "str_value" ):
    print arg is "str_value"

defaultKeepsIdentity()
```

  This now prints "True" as it does with CPython. The solution is actually a general code optimization, see below.

- Usage of `unicode` built-in with more than one argument could corrupt the encoding argument string.

  An implementation error of the `unicode` was releasing references to arguments converted to default encoding, which could corrupt it.

- Assigning Python3 function annotations could cause a segmentation fault.

New Optimization

- Improved propagation of exception raise statements, eliminating more code. They are now also propagated from all kinds of expressions. Previously this was more limited. An assertion added will make sure that all raises are propagated. Also finally, raise expressions are converted into raise statements, but without any normalization.

```
# Now optimizing:
raise TypeError, 1/0
# into (minus normalization):
raise ZeroDivisionError, "integer division or modulo by zero"

# Now optimizing:
(1/0).something
# into (minus normalization):
raise ZeroDivisionError, "integer division or modulo by zero"

# Now optimizing:
function( a, 1/0 ).something
# into (minus normalization), notice the side effects of first checking
# function and a as names to be defined, these may be removed only if
# they can be demonstrated to have no effect.
function
a
raise ZeroDivisionError, "integer division or modulo by zero"
```

There is more examples, where the raise propagation is new, but you get the idea.

- Conditional expression nodes are now optimized according to the truth value of the condition, and not only for compile time constants. This covers e.g. container creations, and other things.

```
# This was already optimized, as it's a compile time constant.
a if ( "a", ) else b
a if True else b

# These are now optimized, as their truth value is known.
a if ( c, ) else b
a if not (c, ) else b
```

  This is simply taking advantage of infrastructure that now exists. Each node kind can overload "getTruthValue" and benefit from it. Help would be welcome to review which ones can be added.

- Function creations only have side effects, when their defaults or annotations (Python3) do. This allows to remove them entirely, should they be found to be unused.

- Code generation for constants now shares element values used in tuples.

  The general case is currently too complex to solve, but we now make sure constant tuples (as e.g. used in the default value for the compiled function), and string constants share the value. This should reduce memory usage and speed up program start-up.

Cleanups

- Optimization was initially designed around visitors that each did one thing, and did it well. It turns out though, that this approach is unnecessary, and constraint collection, allows for the most consistent results. All remaining optimization has been merged into constraint collection.

- The names of modules containing node classes were harmonized to always be plural. In the beginning, this was used to convey the information that only a single node kind would be contained, but that has long changed, and is unimportant information.

- The class names of nodes were stripped from the "CPython" prefix. Originally the intent was to express strict correlation to CPython, but with increasing amounts of re-formulations, this was not used at all, and it's also not important enough to dominate the class name.

- The re-formulations performed in tree building have moved out of the "Building" module, into names "ReformulationClasses" e.g., so they are easier to locate and review. Helpers for node building are now in a separate module, and generally it's much easier to find the content of interest now.
- Added new re-formulation of `print` statements. The conversion to strings is now made explicit in the node tree.

New Tests

- Added test to cover default value identity.

Organizational

- The upload of Nuitka to PyPI has been repaired.

Summary

The release is mostly a consolidation effort, without much performance progress. The progress towards SSA form matter a lot on the outlook front. Once this is finished, standard compiler algorithms can be added to Nuitka which go beyond the current peephole optimization.

Nuitka Release 0.4.0

This release brings massive progress on all fronts. The big highlight is of course: Full Python3.2 support. With this release, the test suite of CPython3.2 is considered passing when compiled with Nuitka.

Then lots of work on optimization and infrastructure. The major goal of this release was to get in shape for actual optimization. This is also why for the first time, it is tested that some things are indeed compile time optimized to spot regressions easier. And we are having performance diagrams, even if weak ones:

New Features

- Python3.2 is now fully supported.
    - Fully correct `metaclass =` semantics now correctly supported. It had been working somewhat previously, but now all the corner cases are covered too.
    - Keyword only parameters.
    - Annotations of functions return value and their arguments.
    - Exception causes, chaining, automatic deletion of exception handlers `as` values.
    - Added support for starred assigns.
    - Unicode variable names are also supported, although it's of course ugly, to find a way to translate these to C++ ones.

Bug fixes

- Checking compiled code with `instance( some_function, types.FunctionType )` as "zope.interfaces" does, was causing compatibility problems. Now this kind of check passes for compiled functions too. Issue#53
- The frame of modules had an empty locals dictionary, which is not compatible to CPython which puts the globals dictionary there too. Also discussed in Issue#53
- For nested exceptions and interactions with generator objects, the exceptions in "sys.exc_info()" were not always fully compatible. They now are.

- The `range` builtin was not raising exceptions if given arguments appeared to not have side effects, but were still illegal, e.g. `range( [], 1, -1 )` was optimized away if the value was not used.
- Don't crash on imported modules with syntax errors. Instead, the attempted recursion is simply not done.
- Doing a `del` on `__defaults` and `__module__` of compiled functions was crashing. This was noticed by a Python3 test for `__kwdefaults__` that exposed this compiled functions weakness.
- Wasn't detecting duplicate arguments, if one of them was not a plain arguments. Star arguments could collide with normal ones.
- The `__doc__` of classes is now only set, where it was in fact specified. Otherwise it only polluted the name space of `locals()`.
- When `return` from the tried statements of a `try/finally` block, was overridden, by the final block, a reference was leaked. Example code:

```
try:
    return 1
finally:
    return 2
```

- Raising exception instances with value, was leaking references, and not raising the `TypeError` error it is supposed to do.
- When raising with multiple arguments, the evaluation order of them was not enforced, it now is. This fixes a reference leak when raising exceptions, where building the exception was raising an exception.

New Optimization

- Optimizing attribute access to compile time constants for the first time. The old registry had no actual user yet.
- Optimizing subscript and slices for all compile time constants beyond constant values, made easy by using inheritance.
- Built-in references now convert to strings directly, e.g. when used in a print statement. Needed for the testing approach "compiled file contains only prints with constant value".
- Optimizing calls to constant nodes directly into exceptions.
- Optimizing built-in `bool` for arguments with known truth value. This would be creations of tuples, lists, and dictionaries.
- Optimizing `a is b` and `a is not b` based on aliasing interface, which at this time effectively is limited to telling that `a is a` is true and `a is not a` is false, but this will expand.
- Added support for optimizing `hasattr`, `getattr`, and `setattr` built-ins as well. The `hasattr` was needed for the `class` re-formulation of Python3 anyway.
- Optimizing `getattr` with string argument and no default to simple attribute access.
- Added support for optimizing `isinstance` built-in.
- Was handling "BreakException" and "ContinueException" in all loops that used `break` or `continue` instead of only where necessary.
- When catching "ReturnValueException", was raising an exception where a normal return was sufficient. Raising them now only where needed, which also means, function need not catch them ever.

Cleanups

- The handling of classes for Python2 and Python3 have been re-formulated in Python more completely.
    - The calling of the determined "metaclass" is now in the node tree, so this call may possible to inline in the future. This eliminated some static C++ code.
    - Passing of values into dictionary creation function is no longer using hard coded special parameters, but temporary variables can now have closure references, making this normal and visible to the optimization.
    - Class dictionary creation functions are therefore no longer as special as they used to be.
    - There is no class creation node anymore, it's merely a call to `type` or the metaclass detected.
- Re-formulated complex calls through helper functions that process the star list and dict arguments and do merges, checks, etc.
    - Moves much C++ code into the node tree visibility.
    - Will allow optimization to eliminate checks and to compile time merge, once inline functions and loop unrolling are supported.
- Added "return None" to function bodies without a an aborting statement at the end, and removed the hard coded fallback from function templates. Makes it explicit in the node tree and available for optimization.
- Merged C++ classes for frame exception keeper with frame guards.
    - The exception is now saved in the compiled frame object, making it potentially more compatible to start with.
    - Aligned module and function frame guard usage, now using the same class.
    - There is now a clear difference in the frame guard classes. One is for generators and one is for functions, allowing to implement their different exception behavior there.
- The optimization registries for calls, subscripts, slices, and attributes have been replaced with attaching them to nodes.
    - The ensuing circular dependency has been resolved by more local imports for created nodes.
    - The package "nuitka.transform.optimization.registries" is no more.
    - New per node methods "computeNodeCall", "computeNodeSubscript", etc. dispatch the optimization process to the nodes directly.
- Use the standard frame guard code generation for modules too.
    - Added a variant "once", that avoids caching of frames entirely.
- The variable closure taking has been cleaned up.
    - Stages are now properly numbered.
    - Python3 only stage is not executed for Python2 anymore.
    - Added comments explaining things a bit better.
    - Now an early step done directly after building a tree.
- The special code generation used for unpacking from iterators and catching "StopIteration" was cleaned up.
    - Now uses template, Generator functions, and proper identifiers.
- The `return` statements in generators are now re-formulated into `raise StopIteration` for generators, because that's what they really are. Allowed to remove special handling of `return` nodes in generators.

- The specialty of CPython2.6 yielding non-None values of lambda generators, was so far implemented in code generation. This was moved to tree building as a re-formulation, making it subject to normal optimization.
- Mangling of attribute names in functions contained in classes, has been moved into the early tree building. So far it was done during code generation, making it invisible to the optimization stages.
- Removed tags attribute from node classes. This was once intended to make up for non-inheritance of similar node kinds, but since we have function references, the structure got so clean, it's no more needed.
- Introduced new package `nuitka.tree`, where the building of node trees, and operations on them live, as well as recursion and variable closure.
- Removed `nuitka.transform` and move its former children `nuitka.optimization` and `nuitka.finalization` one level up. The deeply nested structure turned out to have no advantage.
- Checks for Python version was sometimes "> 300", where of course ">= 300" is the only thing that makes sense.
- Split out helper code for exception raising from the handling of exception objects.

New Tests

- The complete CPython3.2 test suite was adapted (no `__code__`, no `__closure__`, etc.) and is now passing, but only without "--debug", because otherwise some of the generated C++ triggers (harmless) warnings.
- Added new test suite designed to prove that expressions that are known to be compile time constant are indeed so. This works using the XML output done with "--dump-xml" and then searching it to only have print statements with constant values.
- Added new basic CPython3.2 test "Functions32" and "ParameterErrors32" to cover keyword only parameter handling.
- Added tests to cover generator object and exception interactions.
- Added tests to cover `try/finally` and `return` in one or both branches correctly handling the references.
- Added tests to cover evaluation order of arguments when raising exceptions.

Organizational

- Changed my email from GMX over to Gmail, the old one will still continue to work. Updated the copyright notices accordingly.
- Uploaded Nuitka to PyPI as well.

Summary

This release marks a milestone. The support of Python3 is here. The re-formulation of complex calls, and the code generation improvements are quite huge. More re-formulation could be done for argument parsing, but generally this is now mostly complete.

The 0.3.x series had a lot releases. Many of which brought progress with re-formulations that aimed at making optimization easier or possible. Sometimes small things like making "return None" explicit. Sometimes bigger things, like making class creations normal functions, or getting rid of `or` and `and`. All of this was important ground work, to make sure, that optimization doesn't deal with complex stuff.

So, the 0.4.x series begins with this. The focus from now on can be almost purely optimization. This release contains already some of it, with frames being optimized away, with the assignment keepers from

the `or` and `and` re-formulation being optimized away. This will be about achieving goals from the "ctypes" plan as discussed in the developer manual.

Also the performance page will be expanded with more benchmarks and diagrams as I go forward. I have finally given up on "codespeed", and do my own diagrams.

Nuitka Release 0.3.25

This release brings about changes on all fronts, bug fixes, new features. Also very importantly Nuitka no longer uses C++11 for its code, but mere C++03. There is new re-formulation work, and re-factoring of functions.

But the most important part is this: Mercurial unit tests are working. Nearly. With the usual disclaimer of me being wrong, all remaining errors are errors of the test, or minor things. Hope is that these unit tests can be added as release tests to Nuitka. And once that is done, the next big Python application can come.

Bug fixes

- Local variables were released when an exception was raised that escaped the local function. They should only be released, after another exception was raised somewhere. Issue#39.
- Identifiers of nested tuples and lists could collide.

```
a = ( ( 1, 2 ), 3 )
b = ( ( 1, ), 2, 3 )
```

  Both tuples had the same name previously, not the end of the tuple is marked too. Fixed in 0.3.24.1 already.

- The `__name__` when used read-only in modules in packages was optimized to a string value that didn't contain the package name.
- Exceptions set when entering compiled functions were unset at function exit.

New Features

- Compiled frames support. Before, Nuitka was creating frames with the standard CPython C/API functions, and tried its best to cache them. This involved some difficulties, but as it turns out, it is actually possible to instead provide a compatible type of our own, that we have full control over.

  This will become the base of enhanced compatibility. Keeping references to local variables attached to exception tracebacks is something we may be able to solve now.

- Enhanced Python3 support, added support for `nonlocal` declarations and many small corrections for it.
- Writable "__defaults__" attribute for compiled functions, actually changes the default value used at call time. Not supported is changing the amount of default parameters.

Cleanups

- Keep the functions along with the module and added "FunctionRef" node kind to point to them.
- Reformulated `or` and `and` operators with the conditional expression construct which makes the "short-circuit" branch.
- Define "self" as compiled function object instead of pointer to context object, making it possible to access it.

- Removed "OverflowCheck" module and its usage, avoids one useless scan per function to determine the need for "locals dictionary".
- Make "compileTree" of "MainControl" module to only do what the name says and moved the rest out, making the top level control clearer.
- Don't export module entry points when building executable and not modules. These exports cause MinGW and MSVC compilers to create export libraries.

New Optimization

- More efficient code for conditional expressions in conditions:

```
 if a if b else c

See above, this code is now the typical pattern for each ``or`` and ``and``, so this was
much needed now.
```

Organizational

- The remaining uses of C++11 have been removed. Code generated with Nuitka and complementary C++ code now compile with standard C++03 compilers. This lowers the Nuitka requirements and enables at least g++ 4.4 to work with Nuitka.
- The usages of the GNU extension operation `a ?: b` have replaced with standard C++ constructs. This is needed to support MSVC which doesn't have this.
- Added examples for the typical use cases to the User Manual.
- The "compare_with_cpython" script has gained an option to immediately remove the Nuitka outputs (build directory and binary) if successful. Also the temporary files are now put under "/var/tmp" if available.
- Debian package improvements, registering with "doc-base" the User Manual so it is easier to discover. Also suggest "mingw32" package which provides the cross compiler to Windows.
- Partial support for MSVC (Visual Studio 2008 to be exact, the version that works with CPython2.6 and CPython2.7).

  All basic tests that do not use generators are working now, but those will currently cause crashes.
- Renamed the `--g++-only` option to `--c++-only`.

  The old name is no longer correct after clang and MSVC have gained support, and it could be misunderstood to influence compiler selection, rather than causing the C++ source code to not be updated, so manual changes will the used. This solves Issue#47.
- Catch exceptions for `continue`, `break`, and `return` only where needed for `try/finally` and loop constructs.

New Tests

- Added CPython3.2 test suite as "tests/CPython32" from 3.2.3 and run it with CPython2.7 to check that Nuitka gives compatible error messages. It is not expected to pass yet on Python3.2, but work will be done towards this goal.
- Make CPython2.7 test suite runner also execute the generated "doctest" modules.
- Enabled tests for default parameters and their reference counts.

Summary

This release marks an important point. The compiled frames are exciting new technology, that will allow even better integration with CPython, while improving speed. Lowering the requirements to C++03 means, we will become usable on Android and with MSVC, which will make adoption of Nuitka on Windows easier for many.

Structurally the outstanding part is the function as references cleanup. This was a blocker for value propagation, because now functions references can be copied, whereas previously this was duplicating the whole function body, which didn't work, and wasn't acceptable. Now, work can resume in this domain.

Also very exciting when it comes to optimization is the remove of special code for `or` and `and` operators, as these are now only mere conditional expressions. Again, this will make value propagation easier with two special cases less.

And then of course, with Mercurial unit tests running compiled with Nuitka, an important milestone has been hit.

For a while now, the focus will be on completing Python3 support, XML based optimization regression tests, benchmarks, and other open ends. Once that is done, and more certainty about Mercurial tests support, I may call it a 0.4 and start with local type inference for actual speed gains.

Nuitka Release 0.3.24

This release contains progress on many fronts, except performance.

The extended coverage from running the CPython 2.7 and CPython 3.2 (partially) test suites shows in a couple of bug fixes and general improvements in compatibility.

Then there is a promised new feature that allows to compile whole packages.

Also there is more Python3 compatibility, the CPython 3.2 test suite now succeeds up to "test_builtin.py", where it finds that `str` doesn't support the new parameters it has gained, future releases will improve on this.

And then of course, more re-formulation work, in this case, class definitions are now mere simple functions. This and later function references, is the important and only progress towards type inference.

Bug fixes

- The compiled method type can now be used with `copy` module. That means, instances with methods can now be copied too. Issue#40. Fixed in 0.3.23.1 already.
- The `assert` statement as of Python2.7 creates the `AssertionError` object from a given value immediately, instead of delayed as it was with Python2.6. This makes a difference for the form with 2 arguments, and if the value is a tuple. Issue#41. Fixed in 0.3.23.1 already.
- Sets written like this didn't work unless they were predicted at compile time:

  ```
  { value }
  ```

  This apparently rarely used Python2.7 syntax didn't have code generation yet and crashed the compiler. Issue#42. Fixed in 0.3.23.1 already.
- For Python2, the default encoding for source files is `ascii`, and it is now enforced by Nuitka as well, with the same `SyntaxError`.
- Corner cases of `exec` statements with nested functions now give proper `SyntaxError` exceptions under Python2.
- The `exec` statement with a tuple of length 1 as argument, now also gives a `TypeError` exception under Python2.

- For Python2, the `del` of a closure variable is a `SyntaxError`.

New Features

- Added support creating compiled packages. If you give Nuitka a directory with an "__init__.py" file, it will compile that package into a ".so" file. Adding the package contents with `--recurse-dir` allows to compile complete packages now. Later there will be a cleaner interface likely, where the later is automatic.
- Added support for providing directories as main programs. It's OK if they contain a "__main__.py" file, then it's used instead, otherwise give compatible error message.
- Added support for optimizing the `super` built-in. It was already working correctly, but not optimized on CPython2. But for CPython3, the variant without any arguments required dedicated code.
- Added support for optimizing the `unicode` built-in under Python2. It was already working, but will become the basis for the `str` built-in of Python3 in future releases.
- For Python3, lots of compatibility work has been done. The Unicode issues appear to be ironed out now. The `del` of closure variables is allowed and supported now. Built-ins like `ord` and `chr` work more correctly and attributes are now interned strings, so that monkey patching classes works.

Organizational

- Migrated "bin/benchmark.sh" to Python as "misc/run-valgrind.py" and made it a bit more portable that way. Prefers "/var/tmp" if it exists and creates temporary files in a secure manner. Triggered by the Debian "insecure temp file" bug.
- Migrated "bin/make-dependency-graph.sh" to Python as "misc/make-dependency-graph.py" and made a more portable and powerful that way.

  The filtering is done a more robust way. Also it creates temporary files in a secure manner, also triggered by the Debian "insecure temp file" bug.

  And it creates SVG files and no longer PostScript as the first one is more easily rendered these days.
- Removed the "misc/gist" git sub-module, which was previously used by "misc/make-doc.py" to generate HTML from User Manual and Developer Manual. These are now done with Nikola, which is much better at it and it integrates with the web site.
- Lots of formatting improvements to the change log, and manuals:
  - Marking identifiers with better suited ReStructured Text markup.
  - Added links to the bug tracker all Issues.
  - Unified wordings, quotation, across the documents.

Cleanups

- The creation of the class dictionaries is now done with normal function bodies, that only needed to learn how to throw an exception when directly called, instead of returning `NULL`.

  Also the assignment of `__module__` and `__doc__` in these has become visible in the node tree, allowing their proper optimization.

  These re-formulation changes allowed to remove all sorts of special treatment of `class` code in the code generation phase, making things a lot simpler.
- There was still a declaration of `PRINT_ITEMS` and uses of it, but no definition of it.
- Code generation for "main" module and "other" modules are now merged, and no longer special.

- The use of raw strings was found unnecessary and potentially still buggy and has been removed. The dependence on C++11 is getting less and less.

New Tests

- Updated CPython2.6 test suite "tests/CPython26" to 2.6.8, adding tests for recent bug fixes in CPython. No changes to Nuitka were needed in order to pass, which is always good news.
- Added CPython2.7 test suite as "tests/CPython27" from 2.7.3, making it public for the first time. Previously a private copy of some age, with many no longer needed changes had been used by me. Now it is up to par with what was done before for "tests/CPython26", so this pending action is finally done.
- Added test to cover Python2 syntax error of having a function with closure variables nested inside a function that is an overflow function.
- Added test "BuiltinSuper" to cover `super` usage details.
- Added test to cover `del` on nested scope as syntax error.
- Added test to cover `exec` with a tuple argument of length 1.
- Added test to cover `barry_as_FLUFL` future import to work.
- Removed "Unicode" from known error cases for CPython3.2, it's now working.

Summary

This release brought forward the most important remaining re-formulation changes needed for Nuitka. Removing class bodies, makes optimization yet again simpler. Still, making function references, so they can be copied, is missing for value propagation to progress.

Generally, as usual, a focus has been laid on correctness. This is also the first time, I am release with a known bug though: That is Issue#39 which I believe now, may be the root cause of the mercurial tests not yet passing.

The solution will be involved and take a bit of time. It will be about "compiled frames" and be a (invasive) solution. It likely will make Nuitka faster too. But this release includes lots of tiny improvements, for Python3 and also for Python2. So I wanted to get this out now.

As usual, please check it out, and let me know how you fare.

Nuitka Release 0.3.23

This release is the one that completes the Nuitka "sun rise phase".

All of Nuitka is now released under Apache License 2.0 which is a very liberal license, and compatible with basically all Free Software licenses there are. It's only asking to allow integration, of what you send back, and patent grants for the code.

In the first phase of Nuitka development, I wanted to keep control over Nuitka, so it wouldn't repeat mistakes of other projects. This is no longer a concern for me, it's not going to happen anymore.

I would like to thank Debian Legal team, for originally bringing to my attention, that this license will be better suited, than any copyright assignment could be.

Bug fixes

- The compiled functions could not be used with `multiprocessing` or `copy.copy`. Issue#19. Fixed in 0.3.22.1 already.
- In-place operations for slices with not both bounds specified crashed the compiler. Issue#36. Fixed in 0.3.22.1 already.

- Cyclic imports could trigger an endless loop, because module import expressions became the parent of the imported module object. Issue#37. Fixed in 0.3.22.2 already.
- Modules named `proc` or `func` could not be compiled to modules or embedded due to a collision with identifiers of CPython2.7 includes. Issue#38. Fixed in 0.3.22.2 already.

New Features

- The fix for Issue#19 also makes pickling of compiled functions available. As it is the case for non-compiled functions in CPython, no code objects are stored, only names of module level variables.

Organizational

- Using the Apache License 2.0 for all of Nuitka now.
- Speedcenter has been re-activated, but is not yet having a lot of benchmarks yet, subject to change.

New Tests

- Changed the "CPython26" tests to no longer disable the parts that relied on copying of functions to work, as Issue#19 is now supported.
- Extended in-place assignment tests to cover error cases of Issue#36.
- Extended compile library test to also try and compile the path where `numpy` lives. This is apparently another path, where Debian installs some modules, and compiling this would have revealed Issue#36 sooner.

Summary

The release contains bug fixes, and the huge step of changing the license. It is made in preparation to PyCON EU.

Nuitka Release 0.3.22

This release is a continuation of the trend of previous releases, and added more re-formulations of Python that lower the burden on code generation and optimizations.

It also improves Python3 support substantially. In fact this is the first release to not only run itself under Python3, but for Nuitka to *compile itself* with Nuitka under Python3, which previously only worked for Python2. For the common language subset, it's quite fine now.

Bug fixes

- List contractions produced extra entries on the call stack, after they became functions, these are no more existent. That was made possible my making frame stack entries an optional element in the node tree, left out for list contractions.
- Calling a compiled function in an exception handler cleared the exception on return, it no longer does that.
- Reference counter handling with generator `throw` method is now correct.
- A module "builtins" conflicted with the handling of the Python `builtins` module. Those now use different identifiers.

New Features

- New `metaclass` syntax for the `class` statement works, and the old `__metaclass__` attribute is properly ignored.

```
# Metaclass syntax in Python3, illegal in Python2
class X( metaclass = Y ):
    pass
```

```
# Metaclass syntax in Python2, no effect in Python3
class X:
    __metaclass__ = Y
```

> ***Note***
>
> The way to make a use of a metaclass in a portable way, is to create a based class that has it and then inherit f
> isn' it. Surely, the support for `__metaclass__` could still live.
>
> ```
> # For Python2/3 compatible source, we create a base class that has the metaclass
> # used and doesn't require making a choice.
>
> CPythonNodeMetaClassBase = NodeCheckMetaClass( "CPythonNodeMetaClassBase", (object, ), {} )
> ```

- The `--dump-xml` option works with Nuitka running under Python3. This was not previously supported.
- Python3 now also has compatible parameter errors and compatible exception error messages.
- Python3 has changed scope rules for list contractions (assignments don't affect outside values) and this is now respected as well.
- Python3 has gained support for recursive programs and stand alone extension modules, these are now both possible as well.

New Optimization

- Avoid frame stack entries for functions that cannot raise exceptions, i.e. where they would not be used.

  This avoids overhead for the very simple functions. And example of this can be seen here:

```
def simple():
   return 7
```

- Optimize `len` built-in for non-constant, but known length values.

  An example can be seen here:

```
# The range isn't constructed at compile time, but we still know its length.
len( range( 10000000 ) )

# The string isn't constructed at compile time, but we still know its length.
len( "*" * 1000 )

# The tuple isn't constructed, instead it's known length is used, and side effects
```

```
# are maintained.
len( ( a(), b() ) )
```

This new optimizations applies to all kinds of container creations and the `range` built-in initially.

- Optimize conditions for non-constant, but known truth values.

  At this time, known truth values of non-constants means `range` built-in calls with know size and container creations.

  An example can be seen here:

```
if ( a, ):
   print "In Branch"
```

  It's clear, that the tuple will be true, we just need to maintain the side effect, which we do.

- Optimize `or` and `and` operators for known truth values.

  See above for what has known truth values currently. This will be most useful to predict conditions that need not be evaluated at all due to short circuit nature, and to avoid checking against constant values. Previously this could not be optimized, but now it can:

```
# The access and call to "something()" cannot possibly happen
0 and something()

# Can be replaced with "something()", as "1" is true. If it had a side effect, it
# would be maintained.
1 and something()

# The access and call to "something()" cannot possibly happen, the value is already
# decided, it's "1".
1 or something()

# Can be replaced with "something()", as "0" is false. If it had a side effect, it
# would be maintained.
0 or something()
```

- Optimize print arguments to become strings.

  The arguments to `print` statements are now converted to strings at compile time if possible.

```
print 1
```

  becomes:

```
print "1"
```

- Combine print arguments to single ones.

  When multiple strings are printed, these are now combined.

```
print "1+1=", 1+1
```

  becomes:

```
print "1+1= 2"
```

Organizational

- Enhanced Python3 support, enabling support for most basic tests.
- Check files with PyLint in deterministic (alphabetical) order.

Cleanups

- Frame stack entries are now part of the node tree instead of part of the template for every function, generator, class or module.
- The `try/except/else` has been re-formulated to use an indicator variable visible in the node tree, that tells if a handler has been executed or not.
- Side effects are now a dedicated node, used in several optimizations to maintain the effect of an expression with known value.

New Tests

- Expanded and adapted basic tests to work for Python3 as well.
- Added reference count tests for generator functions `throw`, `send`, and `close` methods.
- Cover calling a function with `try/except` in an exception handler twice. No test was previously doing that.

Summary

This release offers enhanced compatibility with Python3, as well as the solution to many structural problems. Calculating lengths of large non-constant values at compile time, is technically a break through, as is avoiding lengthy calculations. The frame guards as nodes is a huge improvement, making that costly operational possible to be optimized away.

There still is more work ahead, before value propagation will be safe enough to enable, but we are seeing the glimpse of it already. Not for long, and looking at numbers will make sense.

Nuitka Release 0.3.21

This releases contains some really major enhancements, all heading towards enabling value propagation inside Nuitka. Assignments of all forms are now all simple and explicit, and as a result, now it will be easy to start tracking them.

Contractions have become functions internally, with statements use temporary variables, complex unpacking statement were reduced to more simple ones, etc.

Also there are the usual few small bug fixes, and a bunch of organizational improvements, that make the release complete.

Bug fixes

- The built-in `next` could causes a program crash when iterating past the end of an iterator. Issue#34. Fixed in 0.3.20.1 already.
- The `set` constants could cause a compiler error, as that type was not considered in the "mutable" check yet. Fixed in 0.3.20.2 already.
- Performance regression. Optimize expression for exception types caught as well again, this was lost in last release.

- Functions that contain `exec`, are supposed to have a writable locals. But when removing that `exec` statement as part of optimizations, this property of the function could get lost.
- The so called "overflow functions" are once again correctly handled. These once were left behind in some refactoring and had not been repaired until now. An overflow function is a nested function with an `exec` or a star import.
- The syntax error for `return` outside of a function, was not given, instead the code returned at run time. Fixed to raise a `SyntaxError` at compile time.

New Optimization

- Avoid `tuple` objects to be created when catching multiple exception types, instead call exception match check function multiple times.
- Removal of dead code following `break`, `continue`, `return`, and `raise`. Code that follows these statements, or conditional statements, where all branches end with it.

> ***Note***
>
> These may not actually occur often in actual code, but future optimizations may produce them more frequently, and their removal may in turn make other possible optimizations.

- Detect module variables as "read only" after all writes have been detected to not be executed as removed. Previously the "read only indicator" was determined only once and then stayed the same.
- Expanded conditional statement optimization to detect cases, where condition is a compile time constant, not just a constant value.
- Optimize away assignments from a variable to the same variable, they have no effect. The potential side effect of accessing the variable is left intact though, so exceptions will be raised still.

> ***Note***
>
> An exception is where `len = len` actually does have an impact, because that variable becomes assignable. The "compile itself" test of Nuitka found that to happen with `long` from the `nuitka.__past__` module.

- Created Python3 variant of quick `unicode` string access, there was no such thing in the CPython C/API, but we make the distinction in the source code, so it makes sense to have it.
- Created an optimized implementation for the built-in `iter` with 2 parameters as well. This allows for slightly more efficient code to be created with regards to reference handling, rather than using the CPython C/API.
- For all types of variable assigned in the generated code, there are now methods that accept already taken references or not, and the code generator picks the optimal variant. This avoids the drop of references, that e.g. the local variable will insist to take.
- Don't use a "context" object for generator functions (and generator expressions) that don't need one. And even if it does to store e.g. the given parameter values, avoid to have a "common context" if there is no closure taken. This avoids useless `malloc` calls and speeds up repeated generator object creation.

Organizational

- Changed the Scons build file database to reside in the build directory as opposed to the current directory, not polluting it anymore. Thanks for the patch go to Michael H Kent, very much appreciated.
- The `--experimental` option is no longer available outside of checkouts of git, and even there not on stable branches (`master`, `hotfix/...`). It only pollutes `--help` output as stable releases have no experimental code options, not even development version will make a difference.
- The binary "bin/Nuitka.py" has been removed from the git repository. It was deprecated a while ago, not part of the distribution and served no good use, as it was a symbolic link only anyway.
- The `--python-version` option is applied at Nuitka start time to re-launch Nuitka with the given Python version, to make sure that the Python run time used for computations and link time Python versions are the same. The allowed values are now checked (2.6, 2.7 and 3.2) and the user gets a nice error with wrong values.
- Added `--keep-pythonpath` alias for `--execute-with-pythonpath` option, probably easier to remember.
- Support `--debug` with clang, so it can also be used to check the generated code for all warnings, and perform assertions. Didn't report anything new.
- The contents environment variable `CXX` determines the default C++ compiler when set, so that checking with `CXX=g++-4.7 nuitka-python ...` has become supported.
- The `check-with-pylint` script now has a real command line option to control the display of "TODO" items.

Cleanups

- Changed complex assignments, i.e. assignments with multiple targets to such using a temporary variable and multiple simple assignments instead.

```
a = b = c
```

```
_tmp = c
b = _tmp
a = _tmp
```

In CPython, when one assignment raises an exception, the whole thing is aborted, so the complexity of having multiple targets is no more needed, now that we have temporary variables in a block.

All that was really needed, was to evaluate the complete source expression only once, but that made code generation contain ugly loops that are no more needed.

- Changed unpacking assignments to use temporary variables. Code like this:

```
a, b = c
```

Is handled more like this:

```
_tmp_iter = iter( c )
_tmp1 = next( _tmp_iter )
_tmp2 = next( _tmp_iter )
if not finished( _tmp_iter ):
```

```
    raise ValueError( "too many values to unpack" )
a = _tmp1
b = _tmp2
```

In reality, not really `next` is used, as it wouldn't raise the correct exception for unpacking, and the `finished` check is more condensed into it.

Generally this cleanup allowed that the `AssignTargetTuple` and associated code generation was removed, and in the future value propagation may optimize these `next` and `iter` calls away where possible. At this time, this is not done yet.

- Exception handlers assign caught exception value through assignment statement.

  Previously the code generated for assigning from the caught exception was not considered part of the handler. It now is the first statement of an exception handler or not present, this way it may be optimized as well.

- Exception handlers now explicitly catch more than one type.

  Catching multiple types worked by merits of the created tuple object working with the Python C/API function called, but that was not explicit at all. Now every handler has a tuple of exceptions it catches, which may only be one, or if None, it's all.

- Contractions are now functions as well.

  Contractions (list, dict, and set) are now re-formulated as function bodies that contain for loops and conditional statements. This allowed to remove a lot of special code that dealt with them and will make these easier to understand for optimization and value propagation.

- Global is handled during tree building.

  Previously the global statement was its own node, which got removed during the optimization phase in a dedicated early optimization that applied its effect, and then removed the node.

  It was determined, that there is no reason to not immediately apply the effect of the global variable and take closure variables and add them to the provider of that `global` statement, allowing to remove the node class.

- Read only module variable detection integrated to constraint collection.

  The detection of read only module variables was so far done as a separate step, which is no more necessary as the constraint collection tracks the usages of module variables anyway, so this separate and slow step could be removed.

New Tests

- Added test to cover order of calls for complex assignments that unpack, to see that they make a fresh iterator for each part of a complex assignment.
- Added test that unpacks in an exception catch. It worked, due to the generic handling of assignment targets by Nuitka, and I didn't even know it can be done, example:

```
try:
    raise ValueError(1,2)
except ValueError as (a,b):
    print "Unpacking caught exception and unpacked", a, b
```

Will assign `a=1` and `b=2`.

- Added test to cover return statements on module level and class level, they both must give syntax errors.

- Cover exceptions from accessing unassigned global names.
- Added syntax test to show that star imports do not allow other names to be imported at the same time as well.
- Python3 is now also running the compile itself test successfully.

Summary

The progress made towards value propagation and type inference is *very* significant, and makes those appears as if they are achievable.

Nuitka Release 0.3.20

This time there are a few bug fixes and some really major cleanups, lots of new optimizations and preparations for more. And then there is a new compiler clang and a new platform supported. MacOS X appears to work mostly, thanks for the patches from Pete Hunt.

Bug fixes

- The use of a local variable name as an expression was not covered and lead to a compiler crash. Totally amazing, but true, nothing in the test suite of CPython covered this. Issue#30. Fixed in release 0.3.19.1 already.
- The use of a closure variable name as an expression was not covered as well. And in this case corrupted the reference count. Issue#31. Fixed in release 0.3.19.1 already.
- The `from x import *` attempted to respect `__all__` but failed to do so. Issue#32. Fixed in release 0.3.19.2 already.
- The `from x import *` didn't give a `SyntaxError` when used on Python3. Fixed in release 0.3.19.2 already.
- The syntax error messages for "global for function argument name" and "duplicate function argument name" are now identical as well.
- Parameter values of generator function could cause compilation errors when used in the closure of list contractions. Fixed.

New Features

- Added support for disabling the console for Windows binaries. Thanks for the patch go to Michael H Kent.
- Enhanced Python3 support for syntax errors, these are now also compatible.
- Support for MacOS X was added.
- Support for using the clang compiler was added, it can be enforced via `--clang` option. Currently this option is mainly intended to allow testing the "MacOS X" support as good as possible under Linux.

New Optimization

- Enhanced all optimizations that previously worked on "constants" to work on "compile time constants" instead. A "compile time constant" can currently also be any form of a built-in name or exception reference. It is intended to expand this in the future.
- Added support for built-ins `bin`, `oct`, and `hex`, which also can be computed at compile time, if their arguments are compile time constant.

- Added support for the `iter` built-in in both forms, one and two arguments. These cannot be computed at compile time, but now will execute faster.
- Added support for the `next` built-in, also in its both forms, one and two arguments. These also cannot be computed at compile time, but now will execute faster as well.
- Added support the the `open` built-in in all its form. We intend for future releases to be able to track file opens for including them into the executable if data files.
- Optimize the `__debug__` built-in constant as well. It cannot be assigned, yet code can determine a mode of operation from it, and apparently some code does. When compiling the mode is decided.
- Optimize the `Ellipsis` built-in constant as well. It falls in the same category as `True`, `False`, `None`, i.e. names of built-in constants that a singletons.
- Added support for anonymous built-in references, i.e. built-ins which have names that are not normally accessible. An example is `type(None)` which is not accessible from anywhere. Other examples of such names are `compiled_method_or_function`. Having these as represented internally, and flagged as "compile time constants", allows the compiler to make more compile time optimizations and to generate more efficient C++ code for it that won't e.g. call the `type` built-in with `None` as an argument.
- All built-in names used in the program are now converted to "built-in name references" in a first step. Unsupported built-ins like e.g. `zip`, for which Nuitka has no own code or understanding yet, remained as "module variables", which made access to them slow, and difficult to recognize.
- Added optimization for module attributes `__file__`, `__doc__` and `__package__` if they are read only. It's the same as `__name__`.
- Added optimization for slices and subscripts of "compile time constant" values. These will play a more important role, once value propagation makes them more frequent.

Organizational

- Created a "change log" from the previous release announcements. It's as ReStructured Text and converted to PDF for the release as well, but I chose not to include that in Debian, because it's so easy to generate the PDF on that yourself.
- The posting of release announcements is now prepared by a script that converts the ReStructured Text to HTML and adds it to Wordpress as a draft posting or updates it, until it's release time. Simple, sweet and elegant.

Cleanups

- Split out the `nuitka.nodes.Nodes` module into many topic nodes, so that there are now `nuitka.nodes.BoolNodes` or `nuitka.nodes.LoopNodes` to host nodes of similar kinds, so that it is now cleaner.
- Split `del` statements into their own node kind, and use much simpler node structures for them. The following blocks are absolutely the same:

```
del a, b.c, d
```

```
del a
del b.c
del d
```

So that's now represented in the node tree. And even more complex looking cases, like this one, also the same:

```
del a, (b.c, d)
```

This one gives a different parse tree, but the same bytecode. And so Nuitka need no longer concern itself with this at all, and can remove the tuple from the parse tree immediately. That makes them easy to handle. As you may have noted already, it also means, there is no way to enforce that two things are deleted or none at all.

- Turned the function and class builder statements into mere assignment statements, where defaults and base classes are handled by wrapping expressions. Previously they are also kind of assignment statements too, which is not needed. Now they were reduced to only handle the `bases` for classes and the `defaults` for functions and make optional.
- Refactored the decorator handling to the tree building stage, presenting them as function calls on "function body expression" or class body expression".

  This allowed to remove the special code for decorators from code generation and C++ templates, making decorations easy subjects for future optimizations, as they practically are now just function calls.

```
@some_classdecorator
class C:
    @staticmethod
    def f():
        pass
```

It's just a different form of writing things. Nothing requires the implementation of decorators, it's just functions calls with function bodies before the assignment.

The following is only similar:

```
class C:
    def f():
        pass

    f = staticmethod( f )

C = some_classdecorator( C )
```

It's only similar, because the assignment to an intermediate value of `C` and `f` is not done, and if an exception was raised by the decoration, that name could persist. For Nuitka, the function and class body, before having a name, are an expression, and so can of course be passed to decorators already.

- The in-place assignments statements are now handled using temporary variable blocks

  Adding support for scoped temporary variables and references to them, it was possible to re-formulate in-place assignments expressions as normal lookups, in-place operation call and then assignment statement. This allowed to remove static templates and will yield even better generated code in the future.
- The for loop used to have has a "source" expression as child, and the iterator over it was only taken at the code generation level, so that step was therefore invisible to optimizations. Moved it to tree building stage instead, where optimizations can work on it then.
- Tree building now generally allows statement sequences to be `None` everywhere, and pass statements are immediately eliminated from them immediately. Empty statement sequences are now forbidden to exist.

- Moved the optimization for `__name__` to compute node of variable references, where it doesn't need anything complex to replace with the constant value if it's only read.
- Added new bases classes and mix-in classes dedicated to expressions, giving a place for some defaults.
- Made the built-in code more reusable.

New Tests

- Added some more diagnostic tests about complex assignment and `del` statements.
- Added syntax test for star import on function level, that must fail on Python3.
- Added syntax test for duplicate argument name.
- Added syntax test for global on a function argument name.

Summary

The decorator and building changes, the assignment changes, and the node cleanups are all very important progress for the type inference work, because they remove special casing the that previously would have been required. Lambdas and functions now really are the same thing right after tree building. The in-place assignments are now merely done using standard assignment code, the built functions and classes are now assigned to names in assignment statements, much *more* consistency there.

Yet, even more work will be needed in the same direction. There may e.g. be work required to cover `with` statements as well. And assignments will become no more complex than unpacking from a temporary variable.

For this release, there is only minimal progress on the Python3 front, despite the syntax support, which is only miniscule progress. The remaining tasks appear all more or less difficult work that I don't want to touch now.

There are still remaining steps, but we can foresee that a release may be done that finally actually does type inference and becomes the effective Python compiler this project is all about.

Nuitka Release 0.3.19

This time there are a few bug fixes, major cleanups, more Python3 support, and even new features. A lot things in this are justifying a new release.

Bug fixes

- The man pages of `nuitka` and `nuitka-python` had no special layout for the option groups and broken whitespace for `--recurse-to` option. Also `--g++-only` was only partially bold. Released as 0.3.18.1 hotfix already.
- The command line length improvement we made to Scons for Windows was not portable to Python2.6. Released as 0.3.18.2 hotfix already.
- Code to detect already considered packages detection was not portable to Windows, for one case, there was still a use of `/` instead of using a `joinpath` call. Released as 0.3.18.3 already.
- A call to the range built-in with no arguments would crash the compiler, see Issue#29. Released as 0.3.18.4 already.
- Compatibility Fix: When rich comparison operators returned false value other `False`, for comparison chains, these would not be used, but `False` instead, see .
- The support for `__import__` didn't cover keyword arguments, these were simply ignored. See Issue#28. Fixed, but no warning is given yet.

New Features

- A new option has been added, one can now specify `--recurse-directory` and Nuitka will attempt to embed these modules even if not obviously imported. This is not yet working perfect yet, but will receive future improvements.
- Added support for the `exec` built-in of Python3, this enables us to run one more basic test, `GlobalStatement.py` with Python3. The test `ExecEval.py` nearly works now.

New Optimization

- The no arguments `range()` call now optimized into the static CPython exception it raises.
- Parts of comparison chains with constant arguments are now optimized away.

Cleanups

- Simplified the `CPythonExpressionComparison` node, it now always has only 2 operands.

  If there are more, the so called "comparison chain", it's done via `and` with assignments to temporary variables, which are expressed by a new node type `CPythonExpressionTempVariableRef`. This allowed to remove `expression_temps` from C++ code templates and generation, reducing the overall complexity.
- When executing a module (`--execute` but not `--exe`), no longer does Nuitka import it into itself, instead a new interpreter is launched with a fresh environment.
- The calls to the variadic `MAKE_TUPLE` were replaced with calls the `MAKE_TUPLExx` (where `xx` is the number of arguments), that are generated on a as-needed basis. This gives more readable code, because no `EVAL_ORDERED_xx` is needed at call site anymore.
- Many node classes have moved to new modules in `nuitka.nodes` and grouped by theme. That makes them more accessible.
- The choosing of the debug python has moved from Scons to Nuitka itself. That way it can respect the `sys.abiflags` and works with Python3.
- The replacing of `.py` in filenames was made more robust. No longer is `str.replace` used, but instead proper means to assure that having `.py` as other parts of the filenames won't be a trouble.
- Module recursion was changed into its own module, instead of being hidden in the optimization that considers import statements.
- As always, some PyLint work, and some minor TODOs were solved.

Organizational

- Added more information to the "Developer Manual", e.g. documenting the tree changes for `assert` to become a conditional statement with a raise statement, etc.
- The Debian package is as of this version verified to be installable and functional on to Ubuntu Natty, Maverick, Oneiric, and Precise.
- Added support to specify the binary under test with a `NUITKA` environment, so the test framework can run with installed version of Nuitka too.
- Made sure the test runners work under Windows as well. Required making them more portable. And a workaround for `os.execl` not propagating exit codes under Windows. See Issue#26 for more information.
- For windows target the MinGW library is now linked statically. That means there is no requirement for MinGW to be in the `PATH` or even installed to execute the binary.

New Tests

- The `basic`, `programs`, `syntax`, and `reflected` were made executable under Windows. Occasionally this meant to make the test runners more portable, or to work around limitations.
- Added test to cover return values of rich comparisons in comparison chains, and order of argument evaluation for comparison chains.
- The `Referencing.py` test was made portable to Python3.
- Cover no arguments `range()` exception as well.
- Added test to demonstrate that `--recurse-directory` actually works. This is using an `__import__` that cannot be predicted at run time (yet).
- The created source package is now tested on pbuilder chroots to be installable and capable of the basic tests, in addition to the full tests during package build time on these chroots. This will make sure, that Nuitka works fine on Ubuntu Natty and doesn't break without notice.

Summary

This releases contains many changes. The "temporary variable ref" and "assignment expression" work is ground breaking. I foresee that it will lead to even more simplifications of code generation in the future, when e.g. in-place assignments can be reduced to assignments to temporary variables and conditional statements.

While there were many improvements related to Windows support and fixing portability bugs, or the Debian package, the real focus is the optimization work, which will ultimately end with "value propagation" working.

These are the real focus. The old comparison chain handling was a big wart. Working, but no way understood by any form of analysis in Nuitka. Now they have a structure which makes their code generation based on semantics and allows for future optimizations to see through them.

Going down this route is an important preparatory step. And there will be more work like this needed. Consider e.g. handling of in-place assignments. With an "assignment expression" to a "temporary variable ref", these become the same as user code using such a variable. There will be more of these to find.

So, that is where the focus is. The release now was mostly aiming at getting involved fixes out. The bug fixed by comparison chain reworking, and the `__import__` related one, were not suitable for hotfix releases, so that is why the 0.3.19 release had to occur now. But with plugin support, with this comparison chain cleanup, with improved Python3 support, and so on, there was plenty of good stuff already, also worth to get out.

Nuitka Release 0.3.18

This is to inform you about the new stable release of Nuitka. This time there are a few bug fixes, and the important step that triggered the release: Nuitka has entered Debian Unstable. So you if want, you will get stable Nuitka releases from now on via `apt-get install nuitka`.

The release cycle was too short to have much focus. It merely includes fixes, which were available as hotfixes, and some additional optimizations and node tree cleanups, as well as source cleanups. But not much else.

Bug fixes

- Conditional statements with both branches empty were not optimized away in all cases, triggering an assertion of code generation. Issue#16. Released as 0.3.17a hotfix already.
- Nuitka was considering directories to contain packages that had no "__init__.py" which could lead to errors when it couldn't find the package later in the compilation process. Released as 0.3.17a hotfix

already.

- When providing `locals()` to `exec` statements, this was not making the `locals()` writable. The logic to detect the case that default value is used (None) and be pessimistic about it, didn't consider the actual value `locals()`. Released as 0.3.17b hotfix already.
- Compatibility Fix: When no defaults are given, CPython uses `None` for `func.func_defaults`, but Nuitka had been using `None`.

New Optimization

- If the condition of assert statements can be predicted, these are now optimized in a static raise or removed.
- For built-in name references, there is now dedicated code to look them up, that doesn't check the module level at all. Currently these are used in only a few cases though.
- Cleaner code is generated for the simple case of `print` statements. This is not only faster code, it's also more readable.

Cleanups

- Removed the `CPythonStatementAssert` node.

  It's not needed, instead at tree building, assert statements are converted to conditional statements with the asserted condition result inverted and a raise statement with `AssertionError` and the assertion argument.

  This allowed to remove code and complexity from the subsequent steps of Nuitka, and enabled existing optimization to work on assert statements as well.
- Moved built-in exception names and built-in names to a new module `nuitka.Builtins` instead of having in other places. This was previously a bit spread-out and misplaced.
- Added cumulative `tags` to node classes for use in checks. Use it annotate which node kinds to visit in e.g. per scope finalization steps. That avoids kinds and class checks.
- New node for built-in name loopups, which allowed to remove tricks played with adding module variable lookups for `staticmethod` when adding them for `__new__` or module variable lookups for `str` when predicting the result of `type( 'a' )`, which was unlikely to cause a problem, but an important TODO item still.

Organizational

- The "Download" page is now finally updated for releases automatically. This closes *Issue#7 <http://bugs.nuitka.net/issue7>* completely. Up to this release, I had to manually edit that page, but I now mastered the art of upload via XMLRCP and a Python script, so that I don't loose as much time with editing, checking it, etc.
- The Debian package is backportable to Ubuntu Natty, Maverick, Oneiric, I expect to make a separate announcement with links to packages.
- Made sure the test runners worth with bare `python2.6` as well.

New Tests

- Added some tests intended for type inference development.

Summary

This releases contains not as much changes as others, mostly because it's the intended base for a Debian upload.

The `exec` fix was detected by continued work on the branch `feature/minimize_CPython26_tests_diff` branch, but that work is now complete.

It is being made pretty (many git rebase iterations) with lots of Issues being added to the bug tracker and referenced for each change. The intention is to have a clean commits repository with the changed made.

But of course, the real excitement is the "type inference" work. It will give a huge boost to Nuitka. With this in place, new benchmarks may make sense. I am working on getting it off the ground, but also to make us more efficient.

So when I learn something. e.g. `assert` is not special, I apply it to the `develop` branch immediately, to keep the differences as small as possible, and to immediately benefit from such improvements.

Nuitka Release 0.3.17

This is to inform you about the new stable release of Nuitka. This time there are a few bug fixes, lots of very important organisational work, and yet again improved compatibility and cleanups. Also huge is the advance in making `--deep` go away and making the recursion of Nuitka controllable, which means a lot for scalability of projects that use a lot of packages that use other packages, because now you can choose which ones to embed and which ones one.

The release cycle had a focus on improving the quality of the test scripts, the packaging, and generally to prepare the work on "type inference" in a new feature branch.

I have also continued to work towards CPython3.2 compatibility, and this version, while not there, supports Python3 with a large subset of the basic tests programs running fine (of course via "2to3" conversion) without trouble. There is still work to do, exceptions don't seem to work fully yet, parameter parsing seems to have changed, etc. but it seems that CPython3.2 is going to work one day.

And there has been a lot of effort, to address the Debian packaging to be cleaner and more complete, addressing issues that prevented it from entering the Debian repository.

Bug fixes

- Fixed the handling of modules and packages of the same name, but with different casing. Problem showed under Windows only. Released as 0.3.16a hotfix already.
- Fixed an error where the command line length of Windows was exceeded when many modules were embedded, Christopher Tott provided a fix for it. Released as 0.3.16a hotfix already.
- Fix, avoid to introduce new variables for where built-in exception references are sufficient. Released as 0.3.16b hotfix already.
- Fix, add the missing `staticmethod` decorator to `__new__` methods before resolving the scopes of variables, this avoids the use of that variable before it was assigned a scope. Released as 0.3.16b hotfix already.

New Features

- Enhanced compatibility again, provide enough `co_varnames` in the code objects, so that slicing them up to `code_object.co_argcount` will work. They are needed by `inspect` module and might be used by some decorators as well.
- New options to control the recursion:

  `--recurse-none` (do not warn about not-done recursions) `--recurse-all` (recurse to all otherwise warned modules) `--recurse-to` (confirm to recurse to those modules)

`--recurse-not-to` (confirm to not recurse to those modules)

New Optimization

- The optimization of constant conditional expressions was not done yet. Added this missing constant propagation case.
- Eliminate near empty statement sequences (only contain a pass statement) in more places, giving a cleaner node structure for many constructs.
- Use the pickle "protocol 2" on CPython2 except for `unicode` strings where it does not work well. It gives a more compressed and binary representation, that is generally more efficient to un-stream as well. Also use the cPickle protocol, the use of `pickle` was not really necessary anymore.

Organizational

- Added a "Developer Manual" to the release. It's incomplete, but it details some of the existing stuff, coding rules, plans for "type inference", etc.
- Improved the `--help` output to use `metavar` where applicable. This makes it more readable for some options.
- Instead of error message, give help output when no module or program file name was given. This makes Nuitka help out more convenient.
- Consistently use `#!/usr/bin/env python` for all scripts, this was previously only done for some of them.
- Ported the PyLint check script to Python as well, enhancing it on the way to check the exit code, and to only output changes things, as well as making the output of warnings for `TODO` items optional.
- All scripts used for testing, PyLint checking, etc. now work with Python3 as well. Most useful on Arch Linux, where it's also already the default for `Python`.
- The help output of Nuitka was polished a lot more. It is now more readable and uses option groups to combine related options together.
- Make the tests run without any dependence on `PATH` to contain the executables of Nuitka. This makes it easier to use.
- Add license texts to 3rd party file that were missing them, apply `licensecheck` results to cleanup Nuitka. Also removed own copyright statement from inline copy of Scons, it had been added by accident only.
- Release the tests that I own as well as the Debian packaging I created under "Apache License 2.0" which is very liberal, meaning every project will be able to use it.
- Don't require copyright assignment for contributions anymore, instead only "Apache License 2.0", the future Nuitka license, so that the code won't be a problem when changing the license of all of Nuitka to that license.
- Give contributors listed in the User Manual an exception to the GPL terms until Nuitka is licensed under "Apache License 2.0" as well.
- Added an `--experimental` option which can be used to control experimental features, like the one currently being added on `feature/ctypes_annotation`, where "type inference" is currently only activated when that option is given. For this stable release, it does nothing.
- Check the static C++ files of Nuitka with `cppcheck` as well. Didn't find anything.
- Arch Linux packages have been contributed, these are linked for download, but the stable package may lag behind a bit.

Cleanups

- Changed `not` boolean operation to become a normal operator. Changed `and` and `or` boolean operators to a new base class, and making their interface more similar to that of operations.
- Added cumulative `tags` to node classes for use in checks. Use it annotate which node kinds to visit in e.g. per scope finalization steps. That avoids kinds and class checks.
- Enhanced the "visitor" interface to provide more kinds of callbacks, enhanced the way "each scope" visiting is achieved by generalizing is as "child has not tag 'closure_taker'" and that for every "node that has tag 'closure_taker'".
- Moved `SyntaxHighlighting` module to `nuitka.gui` package where it belongs.
- More white listing work for imports. As recursion is now the default, and leads to warnings for non-existent modules, the CPython tests gave a lot of good candidates for import errors that were white listed.
- Consistently use `nuitka` in test scripts, as there isn't a `Nuitka.py` on all platforms. The later is scheduled for removal.
- Some more PyLint cleanups.

New Tests

- Make sure the basic tests pass with CPython or else fail the test. This is to prevent false positives, where a test passes, but only because it fails in CPython early on and then does so with Nuitka too. For the syntax tests we make sure they fail.
- The basic tests can now be run with `PYTHON=python3.2` and use `2to3` conversion in that case. Also the currently not passing tests are not run, so the passing tests continue to do so, with this run from the release test script `check-release`.
- Include the syntax tests in release tests as well.
- Changed many existing tests so that they can run under CPython3 too. Of course this is via `2to3` conversion.
- Don't fail if the CPython test suites are not there.

  Currently they remain largely unpublished, and as such are mostly only available to me (exception, `feature/minimize_CPython26_tests_diff` branch references the CPython2.6 tests repository, but that remains work in progress).
- For the compile itself test: Make the presence of the Scons inline copy optional, the Debian package doesn't contain it.
- Also make it more portable, so it runs under Windows too, and allow to choose the Python version to test. Check this test with both CPython2.6 and CPython2.7 not only the default Python.
- Before releasing, test that the created Debian package builds fine in a minimal Debian `unstable` chroot, and passes all the tests included in the package (`basics`, `syntax`, `programs`, `reflected`). Also many other Debian packaging improvements.

Summary

The "git flow" was used again in this release cycle and proved to be useful not only for hotfix, but also for creating the branch `feature/ctypes_annotation` and rebasing it often while things are still flowing.

The few hotfixes didn't require a new release, but the many organizational improvements and the new features did warrant the new release, because of e.g. the much better test handling in this release and the improved recursion control.

The work on Python3 support has slowed down a bit. I mostly only added some bits for compatibility, but generally it has slowed down. I wanted to make sure it doesn't regress by accident, so running with CPython3.2 is now part of the normal release tests.

What's still missing is more "hg" completeness. Only the `co_varnames` work for `inspect` was going in that direction, and this has slowed down. It was more important to make Nuitka's recursion more accessible with the new options, so that was done first.

And of course, the real excitement is the the "type inference" work. It will give a huge boost to Nuitka, and I am happy that it seems to go well. With this in place, new benchmarks may make sense. I am working on getting it off the ground, so other people can work on it too. My idea of `ctypes` native calls may become true sooner than expected. To support that, I would like to add more tools to make sure we discover changes earlier on, checking the XML representations of tests to discover improvements and regressions more clearly.

Nuitka Release 0.3.16

This time there are many bug fixes, some important scalability work, and again improved compatibility and cleanups.

The release cycle had a focus on fixing the bug reports I received. I have also continued to look at CPython3 compatibility, and this is the first version to support Python3 somewhat, at least some of the basic tests programs run (of course via `2to3` conversion) without trouble. I don't know when, but it seems that it's going to work one day.

Also there has an effort to make the Debian packaging cleaner, addressing all kinds of small issues that prevented it from entering the Debian repository. It's still not there, but it's making progress.

Bug fixes

- Fixed a packaging problem for Linux and x64 platform, the new `swapFiber.S` file for the fiber management was not included. Released as 0.3.15a hotfix already.
- Fixed an error where optimization was performed on removed unreachable code, which lead to an error. Released as 0.3.15b hotfix already.
- Fixed an issue with `__import__` and recursion not happening in any case, because when it did, it failed due to not being ported to new internal APIs. Released as 0.3.15c hotfix already.
- Fixed `eval()` and `locals()` to be supported in generator expressions and contractions too. Released as 0.3.15d hotfix already.
- Fixed the Windows batch files `nuitka.bat` and `nuitka-python.bat` to not output the `rem` statements with the copyright header. Released as 0.3.15d hotfix already.
- Fixed re-raise with `raise`, but without a current exception set. Released as 0.3.15e hotfix already.
- Fixed `vars()` call on the module level, needs to be treated as `globals()`. Released as 0.3.15e hotfix already.
- Fix handling of broken new lines in source files. Read the source code in "universal line ending mode". Released as 0.3.15f hotfix already.
- Fixed handling of constant module attribute `__name__` being replaced. Don't replace local variables of the same name too. Released as 0.3.15g hotfix already.
- Fixed assigning to `True`, `False` or `None`. There was this old TODO, and some code has compatibility craft that does it. Released as 0.3.15g hotfix already.
- Fix constant dictionaries not always being recognized as shared. Released as 0.3.15g hotfix already.
- Fix generator function objects to not require a return frame to exist. In finalize cleanup it may not.
- Fixed non-execution of cleanup codes that e.g. flush `sys.stdout`, by adding `Py_Finalize()`.

- Fix `throw()` method of generator expression objects to not check arguments properly.
- Fix missing fallback to subscript operations for slicing with non-indexable objects.
- Fix, in-place subscript operations could fail to apply the update, if the intermediate object was e.g. a list and the handle just not changed by the operation, but e.g. the length did.
- Fix, the future spec was not properly preserving the future division flag.

New Optimization

- The optimization scales now much better, because per-module optimizations only require the module to be reconsidered, but not all modules all the time. With many modules recursed into, this makes a huge difference in compilation time.
- The creation of dictionaries from constants is now also optimized.

New Features

- As a new feature functions now have the `func_defaults` and `__defaults__` attribute. It works only well for non-nested parameters and is not yet fully integrated into the parameter parsing. This improves the compatibility somewhat already though.
- The names `True`, `False` and `None` are now converted to constants only when they are read-only module variables.
- The `PYTHONPATH` variable is now cleared when immediately executing a compiled binary unless `--execute-with-pythonpath` is given, in which case it is preserved. This allows to make sure that a binary is in fact containing everything required.

Organizational

- The help output of Nuitka was polished a lot more. It is now more readable and uses option groups to combine related options together.
- The inline copy of Scons is not checked with PyLint anymore. We of course don't care.
- Program tests are no longer executed in the program directory, so failed module inclusions become immediately obvious.
- The basic tests can now be run with `PYTHON=python3.2` and use `2to3` conversion in that case.

Cleanups

- Moved `tags` to a separate module, make optimizations emit only documented tags, checked against the list of allowed ones.
- The Debian package has seen lots of improvements, to make it "lintian clean", even in pedantic mode. The homepage of Nuitka is listed, a watch file can check for new releases, the git repository and the gitweb are referenced, etc.
- Use `os.path.join` in more of the test code to achieve more Windows portability for them.
- Some more PyLint cleanups.

New Tests

- There is now a `Crasher` test, for tests that crashed Nuitka previously.
- Added a program test where the imported module does a `sys.exit()` and make sure it really doesn't continue after the `SystemExit` exception that creates.

- Cover the type of `__builtins__` in the main program and in imported modules in tests too. It's funny and differs between module and dict in CPython2.
- Cover a final print without newline in the test. Must still receive a newline, which only happens when `Py_Finalize()` is called.
- Added test with functions that makes a `raise` without an exception set.
- Cover the calling of `vars()` on module level too.
- Cover the use of eval in contractions and generator expressions too.
- Cover `func_defaults` and `__default__` attributes for a function too.
- Added test function with two `raise` in an exception handler, so that one becomes dead code and removed without the crash.

Summary

The "git flow" was really great in this release cycle. There were many hotfix releases being made, so that the bugs could be addressed immediately without requiring the overhead of a full release. I believe that this makes Nuitka clearly one of the best supported projects.

This quick turn-around also encourages people to report more bugs, which is only good. And the structure is there to hold it. Of course, the many bug fixes meant that there is not as much new development, but that is not the priority, correctness is.

The work on Python3 is a bit strange. I don't need Python3 at all. I also believe it is that evil project to remove cruft from the Python core and make developers of all relevant Python software, add compatibility cruft to their software instead. Yet, I can't really stop to work on it. It has that appeal of small fixups here and there, and then something else works too.

Python3 work is like when I was first struggling with Nuitka to pass the CPython2 unit tests for a first time. It's fun. And then it finds real actual bugs that apply to CPython2 too. Not doing `Py_Finalize` (but having to), the slice operations shortcomings, the bug of subscript in-place, and so on. There is likely more things hidden, and the earlier Python3 is supported, the more benefit from increased test covered.

What's missing is more "hg" completeness. I think only the `raise` without exception set and the `func_defaults` issue were going into its direction, but it won't be enough yet.

Nuitka Release 0.3.15

This is to inform you about the new stable release of Nuitka. This time again many organizational improvements, some bug fixes, much improved compatibility and cleanups.

This release cycle had a focus on packaging Nuitka for easier consumption, i.e. automatic packaging, making automatic uploads, improvement documentation, and generally cleaning things up, so that Nuitka becomes more compatible and ultimately capable to run the "hg" test suite. It's not there yet, but this is a huge jump for usability of Nuitka and its compatibility, again.

Then lots of changes that make Nuitka approach Python3 support, the generated C++ for at least one large example is compiling with this new release. It won't link, but there will be later releases.

And there is a lot of cleanup going on, geared towards compatibility with line numbers in the frame object.

Bug fixes

- The main module was using `__main__` in tracebacks, but it must be `<module>`. Released as 0.3.14a hotfix already.
- Workaround for "execfile cannot be used as an expression". It wasn't possible to use `execfile` in an expression, only as a statement.

But then there is crazy enough code in e.g. mercurial that uses it in a lambda function, which made the issue more prominent. The fix now allows it to be an expression, except on the class level, which wasn't seen yet.

- The inline copy of Scons was not complete enough to work for "Windows" or with `--windows-target` for cross compile. Fixed.
- Cached frames didn't release the "back" frame, therefore holding variables of these longer than CPython does, which could cause ordering problems. Fixed for increased compatibility.
- Handle "yield outside of function" syntax error in compiled source correctly. This one was giving a Nuitka backtrace, now it gives a `SyntaxError` as it needs to.
- Made syntax/indentation error output absolutely identical to CPython.
- Using the frame objects `f_lineno` may fix endless amounts bugs related to traceback line numbers.

New Features

- Guesses the location of the MinGW compiler under Windows to default install location, so it need not be added to `PATH` environment variable. Removes the need to modify `PATH` environment just for Nuitka to find it.
- Added support for "lambda generators". You don't want to know what it is. Lets just say, it was the last absurd language feature out there, plus that didn't work. It now works perfect.

Organizational

- You can now download a Windows installer and a Debian package that works on Debian Testing, current Ubuntu and Mint Linux.
- New release scripts give us the ability to have hotfix releases as download packages immediately. That means the "git flow" makes even more beneficial to the users.
- Including the generated "README.pdf" in the distribution archives, so it can be read instead of "README.txt". The text file is fairly readable, due to the use of ReStructured Text, but the PDF is even nicer to read, due to e.g. syntax highlighting of the examples.
- Renamed the main binaries to `nuitka` and `nuitka-python`, so that there is no dependency on case sensitive file systems.
- For Windows there are batch files `nuitka.bat` and `nuitka-python.bat` to make Nuitka directly executable without finding the `Python.exe`, which the batch files can tell from their own location.
- There are now man pages of `nuitka` and `nuitka-python` with examples for the most common use cases. They are of course included in the Debian package.
- Don't strip the binary when executing it to analyse compiled binary with `valgrind`. It will give better information that way, without changing the code.

New Optimization

- Implemented `swapcontext` alike (`swapFiber`) for x64 to achieve 8 times speedup for Generators. It doesn't do useless syscalls to preserve signal masks. Now Nuitka is faster at frame switching than CPython on x64, which is already good by design.

Cleanups

- Using the frame objects to store current line of execution avoids the need to store it away in helper code at all. It ought to also help a lot with threading support, and makes Nuitka even more compatible, because now line numbers will be correct even outside tracebacks, but for mere stack frame dumps.
- Moved the `for_return` detection from code generation to tree building where it belongs. Yield statements used as return statements need slightly different code for Python2.6 difference. That solved an old TODO.
- Much Python3 portability work. Sometimes even improving existing code, the Python compiler code had picked up a few points, where the latest Nuitka didn't work with Python3 anymore, when put to actual compile.

  The test covered only syntax, but e.g. meta classes need different code in CPython3, and that's now supported. Also helper code was made portable in more places, but not yet fully. This will need more work.
- Cleaned up uses of debug defines, so they are now more consistent and in one place.
- Some more PyLint cleanups.

New Tests

- The tests are now executed by Python scripts and cover `stderr` output too. Before we only checked `stdout`. This unveiled a bunch of issues Nuitka had, but went unnoticed so far, and triggered e.g. the frame line number improvements.
- Separate syntax tests.
- The scripts to run the tests now are all in pure Python. This means, no more MinGW shell is needed to execute the tests.

Summary

The Debian package, Windows installer, etc. are now automatically updated and uploaded. From here on, there can be such packages for the hotfix releases too.

The exception tracebacks are now correct by design, and better covered.

The generator performance work showed that the approach taken by Nuitka is in fact fast. It was fast on ARM already, but it's nice to see that it's now also fast on x64. Programs using generators will be affected a lot by this.

Overall, this release brings Nuitka closer to usability. Better binary names, man pages, improved documentation, issue tracker, etc. all there now. I am in fact now looking for a sponsor for the Debian package to upload it into Debian directly.

> ***Update***
>
> The upload to Debian happened for 0.3.18 and was done by Yaroslav Halchenko.

What's missing is more "hg" completeness. The frame release issue helped it, but `inspect.getargs()` doesn't work yet, and is a topic for a future release. Won't be easy, as `func_defaults` will be an invasive change too.

Nuitka Release 0.3.14

This is to inform you about the new stable release of Nuitka. This time it contains mostly organisational improvements, some bug fixes, improved compatibility and cleanups.

It is again the result of working towards compilation of a real program (Mercurial). This time, I have added support for proper handling of compiled types by the `inspect` module.

Bug fixes

- Fix for "Missing checks in parameter parsing with star list, star dict and positional arguments". There was whole in the checks for argument counts, now the correct error is given. Fixed in 0.3.13a already.
- The simple slice operations with 2 values, not extended with 3 values, were not applying the correct order for evaluation. Fixed in 0.3.13a already.
- The simple slice operations couldn't handle `None` as the value for lower or upper index. Fixed in 0.3.11a already.
- The in-place simple slice operations evaluated the slice index expressions twice, which could cause problems if they had side effects. Fixed in 0.3.11a already.

New Features

- Run time patching the `inspect` module so it accepts compiled functions, compiled methods, and compiled generator objects. The `test_inspect` test of CPython is nearly working unchanged with this.
- The generator functions didn't have `CO_GENERATOR` set in their code object, setting it made compatible with CPython in this regard too. The inspect module will therefore return correct value for `inspect.isgeneratorfunction()` too.

Optimizations

- Slice indexes that are `None` are now constant propagated as well.
- Slightly more efficient code generation for dual star arg functions, removing useless checks.

Cleanups

- Moved the Scons, static C++ files, and assembler files to new package `nuitka.build` where also now `SconsInterface` module lives.
- Moved the Qt dialog files to `nuitka.gui`
- Moved the "unfreezer" code to its own static C++ file.
- Some PyLint cleanups.

New Tests

- New test `Recursion` to cover recursive functions.
- New test `Inspection` to cover the patching of `inspect` module.
- Cover `execfile` on the class level as well in `ExecEval` test.
- Cover evaluation order of simple slices in `OrderCheck` too.

Organizational

- There is a new issue tracker available under http://bugs.nuitka.net

  Please register and report issues you encounter with Nuitka. I have put all the known issues there and started to use it recently. It's Roundup based like http://bugs.python.org is, so people will find it familiar.

- The `setup.py` is now apparently functional. The source releases for download are made it with, and it appears the binary distributions work too. We may now build a windows installer. It's currently in testing, we will make it available when finished.

Summary

The new source organisation makes packaging Nuitka really easy now. From here, we can likely provide "binary" package of Nuitka soon. A windows installer will be nice.

The patching of `inspect` works wonders for compatibility for those programs that insist on checking types, instead of doing duck typing. The function call problem, was an issue found by the Mercurial test suite.

For the "hg.exe" to pass all of its test suite, more work may be needed, this is the overall goal I am currently striving for. Once real world programs like Mercurial work, we can use these as more meaningful benchmarks and resume work on optimization.

Nuitka Release 0.3.13

This release is mostly the result of working towards compilation of a real programs (Mercurial) and to merge and finalize the frame stack work. Now Nuitka has a correct frame stack at all times, and supports `func_code` and `gi_code` objects, something previously thought to be impossible.

Actually now it's only the "bytecode" objects that won't be there. And not attributes of `func_code` are meaningful yet, but in theory can be supported.

Due to the use of the "git flow" for Nuitka, most of the bugs listed here were already fixed in on the stable release before this release. This time there were 5 such hotfix releases, sometimes fixing multiple bugs.

Bug fixes

- In case of syntax errors in the main program, an exception stack was giving that included Nuitka code. Changed to make the same output as CPython does. Fixed in 0.3.12a already.
- The star import (`from x import *`) didn't work for submodules. Providing `*` as the import list to the respective code allowed to drop the complex lookups we were doing before, and to simply trust CPython C/API to do it correctly. Fixed in 0.3.12 already.
- The absolute import is *not* the default of CPython 2.7 it seems. A local `posix` package shadows the standard library one. Fixed in 0.3.12 already.
- In `--deep` mode, a module may contain a syntax error. This is e.g. true of "PyQt" with `port_v3` included. These files contain Python3 syntax and fail to be imported in Python2, but that is not to be considered an error. These modules are now skipped with a warning. Fixed in 0.3.12b already.
- The code to import modules wasn't using the `__import__` built-in, which prevented `__import__` overriding code to work. Changed import to use the built-in. Fixed in 0.3.12c already.
- The code generated for the `__import__` built-in with constant values was doing relative imports only. It needs to attempt relative and absolut imports. Fixed in 0.3.12c already.
- The code in "__init__.py" believed it was outside of the package, giving problems for package local imports. Fixed in 0.3.12d already.

- It appears that "Scons", which Nuitka uses internally and transparent to you, to execute the compilation and linking tasks, was sometimes not building the binaries or shared libraries, due to a false caching. As a workaround, these are now erased before doing the build. Fixed in 0.3.12d already.
- The use of `in` and `not in` in comparison chains (e.g. `a < b < c` is one), wasn't supported yet. The use of these in comparison chains `a in b in c` is very strange.

  Only in the `test_grammar.py` it was ever used I believe. Anyway, it's supported now, solving this TODO and reducing the difference. Fixed in 0.3.12e already.
- The order of evaluation for `in` and `not in` operators wasn't enforced in a portable way. Now it is correct on "ARM" too. Fixed in 0.3.12e already.

New Optimization

- The built-ins `GeneratorExit` and `StopIteration` are optimized to their Python C/API names where possible as well.

Cleanups

- The `__file__` attribute of modules was the relative filename, but for absolute filenames these become a horrible mess at least on Linux.
- Added assertion helpers for sane frame and code objects and use them.
- Make use of `assertObject` in more places.
- Instead of using `os.path.sep` all over, added a helper `Utils.joinpath` that hides this and using `os.path.join`. This gives more readable code.
- Added traces to the "unfreezer" guarded by a define. Helpful in analyzing import problems.
- Some PyLint cleanups removing dead code, unused variables, useless pass statement, etc.

New Tests

- New tests to cover `SyntaxError` and `IndentationError` from `--deep` imports and in main program.
- New test to cover evaluation order of `in` and `not in` comparisons.
- New test to cover package local imports made by the "__init__.py" of the package.

Organizational

- Drop "compile_itself.sh" in favor of the new "compile_itself.py", because the later is more portable.
- The logging output is now nicer, and for failed recursions, outputs the line that is having the problem.

Summary

The frame stack work and the `func_code` are big for compatibility.

The `func_code` was also needed for "hg" to work. For Mercurial to pass all of its test suite, more work will be needed, esp. the `inspect` module needs to be run-time patched to accept compiled functions and generators too.

Once real world programs like Mercurial work, we can use these as more meaningful benchmarks and resume work on optimization.

Nuitka Release 0.3.12

This is to inform you about the new release of Nuitka many bug fixes, and substantial improvements especially in the organizational area. There is a new User Manual (PDF), with much improved content, a `sys.meta_path` based import mechanism for `--deep` mode, git flow goodness.

This release is generally also the result of working towards compilation of a real programs (Mercurial) and to get things work more nicely on Windows by default. Thanks go to Liu Zhenhai for helping me with this goal.

Due to the use of the "git flow", most of the bugs listed here were already fixed in on the stable release before this release. And there were many of these.

Bug fixes

- The order of evaluation for base classes and class dictionaries was not enforced.

  Apparently nothing in the CPython test suite did that, I only noticed during debugging that Nuitka gave a different error than CPython did, for a class that had an undefined base class, because both class body and base classes were giving an error. Fixed in 0.3.11a already.

- Method objects didn't hold a reference to the used class.

  The effect was only noticed when `--python-debug` was used, i.e. the debug version of Python linked, because then the garbage collector makes searches. Fixed in 0.3.11b already.

- Set `sys.executable` on Linux as well. On Debian it is otherwise `/usr/bin/python` which might be a different version of Python entirely. Fixed in 0.3.11c already.

- Embedded modules inside a package could hide package variables of the same name. Learned during PyCON DE about this corner case. Fixed in 0.3.11d already.

- Packages could be duplicated internally. This had no effect on generated code other than appearing twice in the list if frozen modules. Fixed in 0.3.11d already.

- When embedding modules from outside current directory, the look-up failed. The embedding only ever worked for the compile itself and programs test cases, because they are all in the current directory then. Fixed in 0.3.11e already.

- The check for ARM target broke Windows support in the Scons file. Fixed in 0.3.11f already.

- The star import from external modules failed with an error in `--deep` mode. Fixed in 0.3.11g already.

- Modules with a parent package could cause a problem under some circumstances. Fixed in 0.3.11h already.

- One call variant, with both list and dict star arguments and keyword arguments, but no positional parameters, didn't have the required C++ helper function implemented. Fixed in 0.3.11h already.

- The detection of the CPU core count was broken on my hexacore at least. Gave 36 instead of 6, which is a problem for large programs. Fixed in 0.3.11h already.

- The inline copy of Scons didn't really work on Windows, which was sad, because we added it to simplify installation on Windows precisely because of this.

- Cleaning up the build directory from old sources and object files wasn't portable to Windows and therefore wasn't effective there.

- From imports where part of the imported were found modules and parts were not, didn't work. Solved by the feature branch `meta_path_import` that was merged for this release.

- Newer MinGW gave warnings about the default visibility not being possible to apply to class members. Fixed by not setting this default visibility anymore on Windows.

- The `sys.executable` gave warnings on Windows because of backslashes in the path. Using a raw string to prevent such problems.
- The standard library path was hard coded. Changed to run time detection.

Cleanups

- Version checks on Python runtime now use a new define `PYTHON_VERSION` that makes it easier. I don't like `PY_VERSION_HEX`, because it is so unreadable. Makes some of the checks a lot more safe.
- The `sys.meta_path` based import from the `meta_path_import` feature branch allowed the cleanup the way importing is done. It's a lot less code now.
- Removed some unused code. We will aim at making Nuitka the tool to detect dead code really.
- Moved `nuitka.Nodes` to `nuitka.nodes.Nodes`, that is what the package is intended for, the split will come later.

New Tests

- New tests for import variants that previously didn't work: Mixed imports. Imports from a package one level up. Modules hidden by a package variable, etc.
- Added test of function call variant that had no test previously. Only found it when compiling "hg". Amazing how nothing in my tests, CPython tests, etc. used it.
- Added test to cover the partial success of import statements.
- Added test to cover evaluation order of class definitions.

Organizational

- Migrated the "README.txt" from org-mode to ReStructured Text, which allows for a more readable document, and to generate a nice User Manual in PDF form.
- The amount of information in "README.txt" was increased, with many more subjects are now covered, e.g. "git flow" and how to join Nuitka development. It's also impressive to see what code blocks and syntax highlighting can do for readability.
- The Nuitka git repository has seen multiple hotfixes.

  These allowed to publish bug fixes immediately after they were made, and avoided the need for a new release just to get these out. This really saves me a lot of time too, because I can postpone releasing the new version until it makes sense because of other things.
- Then there was a feature branch `meta_path_import` that lived until being merged to `develop` to improve the import code, which is now released on `master` as stable. Getting that feature right took a while.
- And there is the feature branch `minimize_CPython26_tests_diff` which has some success already in documenting the required changes to the "CPython26" test suite and in reducing the amount of differences, while doing it. We have a frame stack working there, albeit in too ugly code form.
- The release archives are now built using `setuptools`. You can now also download a zip file, which is probably more Windows friendly. The intention is to work on that to make `setup.py` produce a Nuitka install that won't rely on any environment variables at all. Right now `setup.py` won't even allow any other options than `sdist` to be given.
- Ported "compile_itself.sh" to "compile_itself.py", i.e. ported it to Python. This way, we can execute it easily on Windows too, where it currently still fails. Replacing `diff`, `rm -rf`, etc. is a challenge, but

it reduces the dependency on MSYS tools on Windows.

- The compilation of standard library is disabled by default, but `site` or `dist` packages are now embedded. To include even standard library, there is a `--really-deep` option that has to be given in addition to `--deep`, which forces this.

Summary

Again, huge progress. The improved import mechanism is very beautiful. It appears that little is missing to compile real world programs like "hg" with Nuitka. The next release cycle will focus on that and continue to improve the Windows support which appears to have some issues.

Nuitka Release 0.3.11

This is to inform you about the new release of Nuitka with some bug fixes and portability work.

This release is generally cleaning up things, and makes Nuitka portable to ARM Linux. I used to host the Nuitka homepage on that machine, but now that it's no longer so, I can run heavy compile jobs on it. To my surprise, it found many portability problems. So I chose to fix that first, the result being that Nuitka now works on ARM Linux too.

Bug fixes

- The order of slice expressions was not correct on x86 as well, and I found that with new tests only. So the porting to ARM revealed a bug category, I previously didn't consider.
- The use of `linux2` in the Scons file is potentially incompatible with Linux 3.0, although it seems that at least on Debian the `sys.platform` was changed back to `linux2`. Anyway, it's probably best to allow just anything that starts with `linux` these days.
- The `print` statement worked like a `print` function, i.e. it first evaluated all printed expressions, and did the output only then. That is incompatible in case of exceptions, where partial outputs need to be done, and so that got fixed.

New Optimization

- Function calls now each have a dedicated helper function, avoiding in some cases unnecessary work. We will may build further on this and inline `PyObject_Call` differently for the special cases.

Cleanups

- Moved many C++ helper declarations and inline implementations to dedicated header files for better organisation.
- Some dependencies were removed and consolidated to make the dependency graph sane.
- Multiple decorators were in reverse order in the node tree. The code generation reversed it back, so no bug, yet that was a distorted tree.

  Finding this came from the ARM work, because the "reversal" was in fact just the argument evaluation order of C++ under x86/x64, but on ARM that broke. Correcting it highlighted this issue.
- The deletion of slices, was not using `Py_ssize` for indexes, disallowing some kinds of optimizations, so that was harmonized.
- The function call code generation got a general overhaul. It is now more consistent, has more helpers available, and creates more readable code.
- PyLint is again happier than ever.

New Tests

- There is a new basic test `OrderChecks` that covers the order of expression evaluation. These problems were otherwise very hard to detect, and in some cases not previously covered at all.
- Executing Nuitka with Python3 (it won't produce correct Python3 C/API code) is now part of the release tests, so non-portable code of Nuitka gets caught.

Organizational

- Support for ARM Linux. I will make a separate posting on the challenges of this. Suffice to say now, that C++ leaves way too much things unspecified.
- The Nuitka git repository now uses "git flow". The new git policy will be detailed in another separate posting.
- There is an unstable `develop` branch in which the development occurs. For this release ca. 40 commits were done to this branch, before merging it. I am also doing more fine grained commits now.
- Unlike previously, there is `master` branch for the stable release.
- There is a script "make-dependency-graph.sh" (now called "make-dependency-graph.py") to produce a dependency graphs of Nuitka. I detected a couple of strange things through this.
- The Python3 `__pycache__` directories get removed too by the cleanup script.

Numbers

We only have "PyStone" now, and on a new machine, so the numbers cannot be compared to previous releases:

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.48
This machine benchmarks at 104167 pystones/second
```

Nuitka 0.3.11 (driven by python 2.6):

```
Pystone(1.1) time for 50000 passes = 0.19
This machine benchmarks at 263158 pystones/second
```

So this a speedup factor of 258%, last time on another machine it was 240%. Yet it only proves that the generated and compiled are more efficient than bytecode, but Nuitka doesn't yet do the relevant optimizations. Only once it does, the factor will be significantly higher.

Summary

Overall, there is quite some progress. Nuitka is a lot cleaner now, which will help us later only. I wanted to get this out, mostly because of the bug fixes, and of course just in case somebody attempts to use it on ARM.

Nuitka Release 0.3.10

This new release is major milestone 2 work, enhancing practically all areas of Nuitka. The focus was roundup and breaking new grounds with structural optimization enhancements.

Bug fixes

- Exceptions now correctly stack.

  When you catch an exception, there always was the exception set, but calling a new function, and it catching the exception, the values of `sys.exc_info()` didn't get reset after the function returned.

  This was a small difference (of which there are nearly none left now) but one that might effect existing code, which affects code that calls functions in exception handling to check something about it.

  So it's good this is resolved now too. Also because it is difficult to understand, and now it's just like CPython behaves, which means that we don't have to document anything at all about it.

- Using `exec` in generator functions got fixed up. I realized that this wouldn't work while working on other things. It's obscure yes, but it ought to work.
- Lambda generator functions can now be nested and in generator functions. There were some problems here with the allocation of closure variables that got resolved.
- List contractions could not be returned by lambda functions. Also a closure issue.
- When using a mapping for globals to `exec` or `eval` that had a side effect on lookup, it was evident that the lookup was made twice. Correcting this also improves the performance for the normal case.

New Optimization

- Statically raised as well as predicted exceptions are propagated upwards, leading to code and block removal where possible, while maintaining the side effects.

  This is brand new and doesn't do everything possible yet. Most notable, the matching of raised exception to handlers is not yet performed.

- Built-in exception name references and creation of instances of them are now optimized as well, which leads to faster exception raising/catching for these cases.
- More kinds of calls to built-ins are handled, positional parameters are checked and more built-ins are covered.

  Notable is that now checks are performed if you didn't potentially overload e.g. the `len` with your own version in the module. Locally it was always detected already. So it's now also safe.

- All operations and comparisons are now simulated if possible and replaced with their result.
- In the case of predictable true or false conditions, not taken branches are removed.
- Empty branches are now removed from most constructs, leading to sometimes cleaner code generated.

Cleanups

- Removed the lambda body node and replaced it with function body. This is a great win for the split into body and builder. Regular functions and lambda functions now only differ in how the created body is used.
- Large cleanup of the operation/comparison code. There is now only use of a simulator function, which exists for every operator and comparison. This one is then used in a prediction call, shared with the built-in predictions.
- Added a `Tracing` module to avoid future imports of `print_function`, which annoyed me many times by causing syntax failures for when I quickly added a print statement, not noting it must have the braces.
- PyLint is happier than ever.

New Tests

- Enhanced `OverflowFunctions` test to cover even deeper nesting of overflow functions taking closure from each level. While it's not yet working, this makes clearer what will be needed. Even if this code is obscure, I would like to be that correct here.
- Made `Operators` test to cover the `` operator as well.
- Added to `ListContractions` the case where a contraction is returned by a lambda function, but still needs to leak its loop variable.
- Enhanced `GeneratorExpressions` test to cover lambda generators, which is really crazy code:

```
def y():
    yield((yield 1),(yield 2))
```

- Added to `ExecEval` a case where the `exec` is inside a generator, to cover that too.
- Activated the testing of `sys.exc_info()` in `ExceptionRaising` test. This was previously commented out, and now I added stuff to illustrate all of the behaviour of CPython there.
- Enhanced `ComparisonChains` test to demonstrate that the order of evaluations is done right and that side effects are maintained.
- Added `BuiltinOverload` test to show that overloaded built-ins are actually called and not the optimized version. So code like this has to print 2 lines:

```
from __builtin__ import len as _len

def len( x ):
   print x

return _len(x)

print len(range(9))
```

Organizational

- Changed "README.txt" to no longer say that "Scons" is a requirement. Now that it's included (patched up to work with `ctypes` on Windows), we don't have to say that anymore.
- Documented the status of optimizations and added some more ideas.
- There is now an option to dump the node tree after optimization as XML. Not currently use, but is for regression testing, to identify where new optimization and changes have an impact. This make it more feasible to be sure that Nuitka is only becoming better.
- Executable with Python3 again, although it won't do anything, the necessary code changes were done.

Summary

It's nice to see, that I some long standing issues were resolved, and that structural optimization has become almost a reality.

The difficult parts of exception propagation are all in place, now it's only details. With that we can eliminate and predict even more of the stupid code of "pybench" at compile time, achieving more infinite speedups.

Nuitka Release 0.3.9

This is about the new release of Nuitka which some bug fixes and offers a good speed improvement.

This new release is major milestone 2 work, enhancing practically all areas of Nuitka. The main focus was on faster function calls, faster class attributes (not instance), faster unpacking, and more built-ins detected and more thoroughly optimizing them.

Bug fixes

- Exceptions raised inside with statements had references to the exception and traceback leaked.
- On Windows the binaries `sys.executable` pointed to the binary itself instead of the Python interpreter. Changed, because some code uses `sys.executable` to know how to start Python scripts.
- There is a bug (fixed in their repository) related to C++ raw strings and C++ "trigraphs" that affects Nuitka, added a workaround that makes Nuitka not emit "trigraphs" at all.
- The check for mutable constants was erroneous for tuples, which could lead to assuming a tuple with only mutable elements to be not mutable, which is of course wrong.

New Optimization

This time there are so many new optimizations, it makes sense to group them by the subject.

Exceptions

- The code to add a traceback is now our own, which made it possible to use frames that do not contain line numbers and a code object capable of lookups.
- Raising exceptions or adding to tracebacks has been made way faster by reusing a cached frame objects for the task.
- The class used for saving exceptions temporarily (e.g. used in `try/finally` code, or with statement) has been improved so it doesn't make a copy of the exception with a C++ `new` call, but it simply stores the exception properties itself and creates the exception object only on demand, which is more efficient.
- When catching exceptions, the addition of tracebacks is now done without exporting and re-importing the exception to Python, but directly on the exception objects traceback, this avoids a useless round trip.

Function Calls

- Uses of PyObject_Call provide `NULL` as the dictionary, instead of an empty dictionary, which is slightly faster for function calls.
- There are now dedicated variants for complex function calls with `*` and `**` arguments in all forms. These can take advantage of easier cases. For example, a merge with star arguments is only needed if there actually were any of these.
- The check for non-string values in the `**` arguments can now be completely short-cut for the case of a dictionary that has never had a string added. There is now code that detects this case and skips the check, eliminating it as a performance concern.

Parameter Parsing

- Reversed the order in which parameters are checked.

Now the keyword dictionary is iterated first and only then the positional arguments after that is done. This iteration is not only much faster (avoiding repeated lookups for each possible parameter), it also can be more correct, in case the keyword argument is derived from a dictionary and its keys mutate it when being compared.

- Comparing parameter names is now done with a fast path, in which the pointer values are compare first. This can avoid a call to the comparison at all, which has become very likely due to the interning of parameter name strings, see below.
- Added a dedicated call to check for parameter equality with rich equality comparison, which doesn't raise an exception.
- Unpacking of tuples is now using dedicated variants of the normal unpacking code instead of rolling out everything themselves.

Attribute Access

- The class type (in executables, not yet for extension modules) is changed to a faster variant of our own making that doesn't consider the restricted mode a possibility. This avoids very expensive calls, and makes accessing class attributes in compiled code and in non-compiled code faster.
- Access to attributes (but not of instances) got inlined and therefore much faster. Due to other optimizations, a specific step to intern the string used for attribute access is not necessary with Nuitka at all anymore. This made access to attributes about 50% faster which is big of course.

Constants

- The bug for mutable tuples also caused non-mutable tuples to be considered as mutable, which lead to less efficient code.
- The constant creation with the g++ bug worked around, can now use raw strings to create string constants, without resorting to un-pickling them as a work around. This allows us to use `PyString_FromStringAndSize` to create strings again, which is obviously faster, and had not been done, because of the confusion caused by the g++ bug.
- For string constants that are usable as attributes (i.e. match the identifier regular expression), these are now interned, directly after creation. With this, the check for identical value of pointers for parameters has a bigger chance to succeed, and this saves some memory too.
- For empty containers (set, dict, list, tuple) the constants created are now are not unstreamed, but created with the dedicated API calls, saving a bit of code and being less ugly.
- For mutable empty constant access (set, dict, list) the values are no longer made by copying the constant, but instead with the API functions to create new ones. This makes code like `a = []` a tiny bit faster.
- For slice indices the code generation now takes advantage of creating a C++ `Py_ssize_t` from constant value if possible. Before it was converting the integer constant at run time, which was of course wasteful even if not (very) slow.

Iteration

- The creation of iterators got our own code. This avoids a function call and is otherwise only a small gain for anything but sequence iterators. These may be much faster to create now, as it avoids another call and repeated checks.
- The next on iterator got our own code too, which has simpler code flow, because it avoids the double check in case of NULL returned.
- The unpack check got simlar code to the next iterator, it also has simpler code flow now and avoids double checks.

Built-ins

- Added support for the `list`, `tuple`, `dict`, `str`, `float` and `bool` built-ins along with optimizing their use with constant parameter.
- Added support for the `int` and `long` built-ins, based on a new "call spec" object, that detects parameter errors at compile time and raises appropriate exceptions as required, plus it deals with keyword arguments just as well.

  So, to Nuitka it doesn't matter now it you write `int( value )` ``or ``int(x = value)` anymore. The `base` parameter of these built-ins is also supported.

  The use of this call spec mechanism will the expanded, currently it is not applied to the built-ins that take only one parameter. This is a work in progress as is the whole built-ins business as not all the built-ins are covered yet.

Cleanups

- In 0.3.8 per module global classes were introduced, but the `IMPORT_MODULE` kept using the old universal class, this got resolved and the old class is now fully gone.
- Using `assertObject` in more cases, and in more places at all, catches errors earlier on.
- Moved the addition to tracebacks into the `_PythonException` class, where it works directly on the contained traceback. This is cleaner as it no longer requires to export exceptions to Python, just to add a traceback entry.
- Some `PyLint` cleanups were done, reducing the number of reports a bit, but there is still a lot to do.
- Added a `DefaultValueIdentifier` class that encapsulates the access to default values in the parameter parsing more cleanly.
- The module `CodeTemplatesListContractions` was renamed to `CodeTemplatesContractions` to reflect the fact that it deals with all kinds of contractions (also set and dict contractions), not just list contractions.
- Moved the with related template to its own module `CodeTemplatesWith`, so its easier to find.
- The options handling for g++ based compilers was cleaned up, so that g++ 4.6 and MinGW are better supported now.
- Documented more aspects of the Scons build file.
- Some more generated code white space fixes.
- Moved some helpers to dedicated files. There is now `calling.hpp` for function calls, an `importing.cpp` for import related stuff.
- Moved the manifest generation to the scons file, which now produces ready to use executables.

New Tests

- Added a improved version of "pybench" that can cope with the "0 ms" execution time that Nuitka has for some if its sub-tests.
- Reference counting test for with statement was added.
- Micro benchmarks to demonstrate try finally performance when an exception travels through it.
- Micro benchmark for with statement that eats up exceptions raised inside the block.
- Micro benchmarks for the read and write access to class attributes.
- Enhanced `Printing` test to cover the trigraphs constant bug case. Output is required to make the error detectable.

- Enhanced `Constants` test to cover repeated mutation of mutable tuple constants, this covers the bug mentioned.

Organizational

- Added a credits section to the "README.txt" where I give credit to the people who contributed to Nuitka, and the projects it is using. I will make it a separate posting to cite these.
- Documented the requirements on the compiler more clearly, document the fact that we require scons and which version of Python (2.6 or 2.7).
- The is now a codespeed implementation up and running with historical data for up to Nuitka 0.3.8 runs of "PyStone" and with pybench. It will be updated for 0.3.9 once I have the infrastructure in place to do that automatically.
- The cleanup script now also removes .so files.
- The handling of options for g++ got improved, so it's the same for g++ and MinGW compilers, plus adequate errors messages are given, if the compiler version is too low.
- There is now a `--unstriped` option that just keeps the debug information in the file, but doesn't keep the assertions. This will be helpful when looking at generated assembler code from Nuitka to not have the distortions that `--debug` causes (reduced optimization level, assertions, etc.) and instead a clear view.

Nuitka Release 0.3.8

This is to inform you about the new release of Nuitka with some real news and a slight performance increase. The significant news is added "Windows Support". You can now hope to run Nuitka on Windows too and have it produce working executables against either the standard Python distribution or a MinGW compiled Python.

There are still some small things to iron out, and clearly documentation needs to be created, and esp. the DLL hell problem of `msvcr90.dll` vs. `msvcrt.dll`, is not yet fully resolved, but appears to be not as harmful, at least not on native Windows.

I am thanking Khalid Abu Bakr for making this possible. I was surprised to see this happen. I clearly didn't make it easy. He found a good way around `ucontext`, identifier clashes, and a very tricky symbol problems where the CPython library under Windows exports less than under Linux. Thanks a whole lot.

Currently the Windows support is considered experimental and works with MinGW 4.5 or higher only.

Otherwise there have been the usual round of performance improvements and more cleanups. This release is otherwise milestone 2 work only, which will have to continue for some time more.

Bug fixes

- Lambda generators were not fully compatible, their simple form could yield an extra value. The behavior for Python 2.6 and 2.7 is also different and Nuitka now mimics both correctly, depending on the used Python version
- The given parameter count cited in the error message in case of too many parameters, didn't include the given keyword parameters in the error message.
- There was an `assert False` right after warning about not found modules in the `--deep` mode, which was of course unnecessary.

New Optimization

- When unpacking variables in assignments, the temporary variables are now held in a new temporary class that is designed for the task specifically.

  This avoids the taking of a reference just because the `PyObjectTemporary` destructor insisted on releasing one. The new class `PyObjectTempHolder` hands the existing reference over and releases only in case of exceptions.

- When unpacking variable in for loops, the value from the iterator may be directly assigned, if it's to a variable.

  In general this would be possible for every assignment target that cannot raise, but the infrastructure cannot tell yet, which these would be. This will improve with more milestone 3 work.

- Branches with only `pass` inside are removed, `pass` statements are removed before the code generation stage. This makes it easier to achieve and decide empty branches.

- There is now a global variable class per module. It appears that it is indeed faster to roll out a class per module accessing the `module *` rather than having one class and use a `module **`, which is quite disappointing from the C++ compiler.

- Also `MAKE_LIST` and `MAKE_TUPLE` have gained special cases for the 0 arguments case. Even when the size of the variadic template parameters should be known to the compiler, it seems, it wasn't eliminating the branch, so this was a speedup measured with valgrind.

- Empty tried branches are now replaced when possible with `try/except` statements, `try/finally` is simplified in this case. This gives a cleaner tree structure and less verbose C++ code which the compiler threw away, but was strange to have in the first place.

- In conditions the `or` and `and` were evaluated with Python objects instead of with C++ bool, which was unnecessary overhead.

- List contractions got more clever in how they assign from the iterator value.

  It now uses a `PyObjectTemporary` if it's assigned to multiple values, a `PyObjectTempHolder` if it's only assigned once, to something that could raise, or a `PyObject *` if an exception cannot be raised. This avoids temporary references completely for the common case.

Cleanups

- The `if`, `for`, and `while` statements had always empty `else` nodes which were then also in the generated C++ code as empty branches. No harm to performance, but this got cleaned up.

- Some more generated code white space fixes.

New Tests

- The CPython 2.7 test suite now also has the `doctests` extracted to static tests, which improves test coverage for Nuitka again.

  This was previously only done for CPython 2.6 test suite, but the test suites are different enough to make this useful, e.g. to discover newly changed behavior like with the lambda generators.

- Added Shed Skin 0.7.1 examples as benchmarks, so we can start to compare Nuitka performance in these tests. These will be the focus of numbers for the 0.4.x release series.

- Added a micro benchmark to check unpacking behavior. Some of these are needed to prove that a change is an actual improvement, when its effect can go under in noise of inline vs. no-inline behavior of the C++ compiler.

- Added "pybench" benchmark which reveals that Nuitka is for some things much faster, but there are still fields to work on. This version needed changes to stand the speed of Nuitka. These will be subject of a later posting.

Organizational

- There is now a "tests/benchmarks/micro" directory to contain tiny benchmarks that just look at a single aspect, but have no other meaning, e.g. the "PyStone" extracts fall into this category.
- There is now a `--windows-target` option that attempts a cross-platform build on Linux to Windows executable. This is using "MingGW-cross-env" cross compilation tool chain. It's not yet working fully correctly due to the DLL hell problem with the C runtime. I hope to get this right in subsequent releases.
- The `--execute` option uses wine to execute the binary if it's a cross-compile for windows.
- Native windows build is recognized and handled with MinGW 4.5, the VC++ is not supported yet due to missing C++0x support.
- The basic test suite ran with Windows so far only and some adaptations were necessary. Windows new lines are now ignored in difference check, and addresses under Windows are upper case, small things.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.8 (driven by python 2.6):

```
Pystone(1.1) time for 50000 passes = 0.27
This machine benchmarks at 185185 pystones/second
```

This is a 140% speed increase of 0.3.8 compared to CPython, up from 132% compared to the previous release.

Nuitka Release 0.3.7

This is about the new release with focus on performance and cleanups. It indicates significant progress with the milestone this release series really is about as it adds a `compiled_method` type.

So far functions, generator function, generator expressions were compiled objects, but in the context of classes, functions were wrapped in CPython `instancemethod` objects. The new `compiled_method` is specifically designed for wrapping `compiled_function` and therefore more efficient at it.

Bug fixes

- When using `Python` or `Nuitka.py` to execute some script, the exit code in case of "file not found" was not the same as CPython. It should be 2, not 1.
- The exit code of the created programs (`--deep` mode) in case of an uncaught exception was 0, now it an error exit with value 1, like CPython does it.
- Exception tracebacks created inside `with` statements could contain duplicate lines, this was corrected.

New Optimization

- Global variable assignments now also use `assign0` where no reference exists.

The assignment code for module variables is actually faster if it needs not drop the reference, but clearly the code shouldn't bother to take it on the outside just for that. This variant existed, but wasn't used as much so far.

- The instance method objects are now Nuitka's own compiled type too. This should make things slightly faster by itself.
- Our new compiled method objects support dedicated method parsing code, where `self` is passed directly, allowing to make calls taking a fast path in parameter parsing.

  This avoids allocating/freeing a `tuple` object per method call, while reduced 3% ticks in "PyStone" benchmark, so that's significant.
- Solved a TODO of `BUILTIN_RANGE` to change it to pre-allocating the list in the final size as we normally do everywhere else. This was a tick reduction of 0.4% in "PyStone" benchmark, but the measurement method normalizes on loop speed, so it's not visible in the numbers output.
- Parameter variables cannot possibly be uninitialized at creation and most often they are never subject to a `del` statement. Adding dedicated C++ variable classes gave a big speedup, around 3% of "PyStone" benchmark ticks.
- Some abstract object operations were re-implemented, which allows to avoid function calls e.g. in the `ITERATOR_NEXT` case, this gave a few percent on "PyStone" as well.

Cleanups

- New package `nuitka.codegen` to contain all code generation related stuff, moved `nuitka.templates` to `nuitka.codegen.templates` as part of that.
- Inside the `nuitka.codegen` package the `MainControl` module now longer reaches into `Generator` for simple things, but goes through `CodeGeneration` for everything now.
- The `Generator` module uses almost no tree nodes anymore, but instead gets information passed in function calls. This allows for a cleanup of the interface towards `CodeGeneration`. Gives a cleaner view on the C++ code generation, and generally furthers the goal of other than C++ language backends.
- More "PyLint" work, many of the reported warnings have been addressed, but it's not yet happy.
- Defaults for `yield` and `return` are `None` and these values are now already added (as constants) during tree building so that no such special cases need to be dealt with in `CodeGeneration` and future analysis steps.
- Parameter parsing code has been unified even further, now the whole entry point is generated by one of the function in the new `nuitka.codegen.ParameterParsing` module.
- Split variable, exception, built-in helper classes into separate header files.

New Tests

- The exit codes of CPython execution and Nuitka compiled programs are now compared as well.
- Errors messages of methods are now covered by the `ParameterErrors` test as well.

Organizational

- A new script "benchmark.sh" (now called "run-valgrind.py") script now starts "kcachegrind" to display the valgrind result directly.

  One can now use it to execute a test and inspect valgrind information right away, then improve it. Very useful to discover methods for improvements, test them, then refine some more.

- The "check-release.sh" script needs to unset `NUITKA_EXTRA_OPTIONS` or else the reflection test will trip over the changed output paths.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.7 (driven by python 2.6):

```
Pystone(1.1) time for 50000 passes = 0.28
This machine benchmarks at 178571 pystones/second
```

This is a 132% speed of 0.3.7 compared to CPython, up from 109% compare to the previous release. This is a another small increase, that can be fully attributed to milestone 2 measures, i.e. not analysis, but purely more efficient C++ code generation and the new compiled method type.

One can now safely assume that it is at least twice as fast, but I will try and get the PyPy or Shedskin test suite to run as benchmarks to prove it.

No milestone 3 work in this release. I believe it's best to finish with milestone 2 first, because these are quite universal gains that we should have covered.

Nuitka Release 0.3.6

The major point this for this release is cleanup work, and generally bug fixes, esp. in the field of importing. This release cleans up many small open ends of Nuitka, closing quite a bunch of consistency TODOs, and then aims at cleaner structures internally, so optimization analysis shall become "easy". It is a correctness and framework release, not a performance improvement at all.

Bug fixes

- Imports were not respecting the `level` yet. Code like this was not working, now it is.

  ```
  from .. import something
  ```

- Absolute and relative imports were e.g. both tried all the time, now if you specify absolute or relative imports, it will be attempted in the same way than CPython does. This can make a difference with compatibility.
- Functions with a "locals dict" (using `locals` built-in or `exec` statement) were not 100% compatible in the way the locals dictionary was updated, this got fixed. It seems that directly updating a dict is not what CPython does at all, instead it only pushes things to the dictionary, when it believes it has to. Nuitka now does the same thing, making it faster and more compatible at the same time with these kind of corner cases.
- Nested packages didn't work, they do now. Nuitka itself is now successsfully using nested packages (e.g. `nuitka.transform.optimizations`)

New Features

- The `--lto` option becomes usable. It's not measurably faster immediately, and it requires g++ 4.6 to be available, but then it at least creates smaller binaries and may provide more optimizations in the future.

New Optimization

- Exceptions raised by pre-computed built-ins, unpacking, etc. are now transformed to raising the exception statically.

Cleanups

- There is now a `getVariableForClosure` that a variable provider can use. Before that it guessed from `getVariableForReference` or `getVariableForAssignment` what might be the intention. This makes some corner cases easier.
- Classes, functions and lambdas now also have separate builder and body nodes, which enabled to make getSameScopeNodes() really simple. Either something has children which are all in a new scope or it has them in the same scope.
- Twisted workarounds like `TransitiveProvider` are no longer needed, because class builder and class body were separated.
- New packages `nuitka.transform.optimizations` and `nuitka.transform.finalizations`, where the first was `nuitka.optimizations` before. There is also code in `nuitka.transform` that was previously in a dedicated module. This allowed to move a lot of displaced code.
- `TreeBuilding` now has fast paths for all 3 forms, things that need a "provider", "node", and "source_ref"; things that need "node" and "source_ref"; things that need nothing at all, e.g. pass.
- Variables now avoid building duplicated instances, but instead share one. Better for analysis of them.

New Tests

- The Python 2.7 test suite is no longer run with Python 2.6 as it will just crash with the same exception all the time, there is no `importlib` in 2.6, but every test is using that through test_support.
- Nested packages are now covered with tests too.
- Imports of upper level packages are covered now too.

Organizational

- Updated the "README.txt" with the current plan on optimizations.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.6 (driven by python 2.6):

```
Pystone(1.1) time for 50000 passes = 0.31
This machine benchmarks at 161290 pystones/second
```

This is 109% for 0.3.6, but no change from the previous release. No surprise, because no new effective new optimization means have been implemented. Stay tuned for future release for actual progress.

Nuitka Release 0.3.5

This new release of Nuitka is an overall improvement on many fronts, there is no real focus this time, likely due to the long time it was in the making.

The major points are more optimization work, largely enhanced import handling and another improvement on the performance side. But there are also many bug fixes, more test coverage, usability and compatibility.

Something esp. noteworthy to me and valued is that many important changes were performed or at least triggered by Nicolas Dumazet, who contributed a lot of high quality commits as you can see from the gitweb history. He appears to try and compile Mercurial and Nuitka, and this resulted in important contributions.

Bug fixes

- Nicolas found a reference counting bug with nested parameter calls. Where a function had parameters of the form `a, (b,c)` it could crash. This got fixed and covered with a reference count test.
- Another reference count problem when accessing the locals dictionary was corrected.
- Values `0.0` and `-0.0` were treated as the same. They are not though, they have a different sign that should not get lost.
- Nested contractions didn't work correctly, when the contraction was to iterate over another contraction which needs a closure. The problem was addressing by splitting the building of a contraction from the body of the contraction, so that these are now 2 nodes, making it easy for the closure handling to get things right.
- Global statements in function with local `exec()` would still use the value from the locals dictionary. Nuitka is now compatible to CPython with this too.
- Nicolas fixed problems with modules of the same name inside different packages. We now use the full name including parent package names for code generation and lookups.
- The `__module__` attribute of classes was only set after the class was created. Now it is already available in the class body.
- The `__doc__` attribute of classes was not set at all. Now it is.
- The relative import inside nested packages now works correctly. With Nicolas moving all of Nuitka to a package, the compile itself exposed many weaknesses.
- A local re-raise of an exception didn't have the original line attached but the re-raise statement line.

New Features

- Modules and packages have been unified. Packages can now also have code in "__init__.py" and then it will be executed when the package is imported.
- Nicolas added the ability to create deep output directory structures without having to create them beforehand. This makes `--output-dir=some/deep/path` usable.
- Parallel build by Scons was added as an option and enabled by default, which enhances scalability for `--deep` compilations a lot.
- Nicolas enhanced the CPU count detection used for the parallel build. Turned out that `multithreading.cpu_count()` doesn't give us the number of available cores, so he contributed code to determine that.
- Support for upcoming g++ 4.6 has been added. The use of the new option `--lto` has been been prepared, but right now it appears that the C++ compiler will need more fixes, before we can this

feature with Nuitka.

- The `--display-tree` feature got an overhaul and now displays the node tree along with the source code. It puts the cursor on the line of the node you selected. Unfortunately I cannot get it to work two-way yet. I will ask for help with this in a separate posting as we can really use a "python-qt" expert it seems.
- Added meaningful error messages in the "file not found" case. Previously I just didn't care, but we sort of approach end user usability with this.

New Optimization

- Added optimization for the built-in `range()` which otherwise requires a module and `builtin` module lookup, then parameter parsing. Now this is much faster with Nuitka and small ranges (less than 256 values) are converted to constants directly, avoiding run time overhead entirely.
- Code for re-raise statements now use a simple re-throw of the exception where possible, and only do the hard work where the re-throw is not inside an exception handler.
- Constant folding of operations and comparisons is now performed if the operands are constants.
- Values of some built-ins are pre-computed if the operands are constants.
- The value of module attribute `__name__` is replaced by a constant unless it is assigned to. This is the first sign of upcoming constant propagation, even if only a weak one.
- Conditional statement and/or their branches are eliminated where constant conditions allow it.

Cleanups

- Nicolas moved the Nuitka source code to its own `nuitka` package. That is going to make packaging it a lot easier and allows cleaner code.
- Nicolas introduced a fast path in the tree building which often delegates (or should do that) to a function. This reduced a lot of the dispatching code and highlights more clearly where such is missing right now.
- Together we worked on the line length issues of Nuitka. We agreed on a style and very long lines will vanish from Nuitka with time. Thanks for pushing me there.
- Nicolas also did provide many style fixes and general improvements, e.g. using `PyObjectTemporary` in more places in the C++ code, or not using `str.find` where `x in y` is a better choice.
- The node structure got cleaned up towards the direction that assigments always have an assignment as a child. A function definition, or a class definition, are effectively assignments, and in order to not have to treat this as special cases everywhere, they need to have assignment targets as child nodes.

  Without such changes, optimizations will have to take too many things into account. This is not yet completed.
- Nicolas merged some node tree building functions that previously handled deletion and assigning differently, giving us better code reuse.
- The constants code generation was moved to a `__constants.cpp` where it doesn't make __main__.cpp so much harder to read anymore.
- The module declarations have been moved to their own header files.
- Nicolas cleaned up the scripts used to test Nuitka big time, removing repetitive code and improving the logic. Very much appreciated.
- Nicolas also documented a things in the Nuitka source code or got me to document things that looked strange, but have reasons behind it.

- Nicolas solved the TODO related to built-in module accesses. These will now be way faster than before.
- Nicolas also solved the TODO related to the performance of "locals dict" variable accesses.
- Generator.py no longer contains classes. The Contexts objects are supposed to contain the state, and as such the generator objects never made much sense.
- Also with the help of Scons community, I figured out how to avoid having object files inside the `src` directory of Nuitka. That should also help packaging, now all build products go to the .build directory as they should.
- The vertical white space of the generated C++ got a few cleanups, trailing/leading new line is more consistent now, and there were some assertions added that it doesn't happen.

New Tests

- The CPython 2.6 tests are now also run by CPython 2.7 and the other way around and need to report the same test failure reports, which found a couple of issues.
- Now the test suite is run with and without `--debug` mode.
- Basic tests got extended to cover more topics and catch more issues.
- Program tests got extended to cover code in packages.
- Added more exec scope tests. Currently inlining of exec statements is disabled though, because it requires entirely different rules to be done right, it has been pushed back to the next release.

Organizational

- The `g++-nuitka` script is no more. With the help of the Scons community, this is now performed inside the scons and only once instead of each time for every C++ file.
- When using `--debug`, the generated C++ is compiled with `-Wall` and `-Werror` so that some form of bugs in the generated C++ code will be detected immediately. This found a few issues already.
- There is a new git merge policy in place. Basically it says, that if you submit me a pull request, that I will deal with it before publishing anything new, so you can rely on the current git to provide you a good base to work on. I am doing more frequent pre-releases already and I would like to merge from your git.
- The "README.txt" was updated to reflect current optimization status and plans. There is still a lot to do before constant propagation can work, but this explains things a bit better now. I hope to expand this more and more with time.
- There is now a "misc/clean-up.sh" script that prints the commands to erase all the temporary files sticking around in the source tree. That is for you if you like me, have other directories inside, ignored, that you don't want to delete.
- Then there is now a script that prints all source filenames, so you can more easily open them all in your editor.
- And very important, there is now a "check-release.sh" script that performs all the tests I think should be done before making a release.
- Pylint got more happy with the current Nuitka source. In some places, I added comments where rules should be granted exceptions.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.5 (driven by python 2.6):

```
Pystone(1.1) time for 50000 passes = 0.31
This machine benchmarks at 161290 pystones/second
```

This is 109% for 0.3.5, up from 91% before.

Overall this release is primarily an improvement in the domain of compatibility and contains important bug and feature fixes to the users. The optimization framework only makes a first showing of with the framework to organize them. There is still work to do to migrate optimizations previously present

It will take more time before we will see effect from these. I believe that even more cleanups of `TreeBuilding`, `Nodes` and `CodeGeneration` will be required, before everything is in place for the big jump in performance numbers. But still, passing 100% feels good. Time to rejoice.

Nuitka Release 0.3.4

This new release of Nuitka has a focus on re-organizing the Nuitka generated source code and a modest improvement on the performance side.

For a long time now, Nuitka has generated a single C++ file and asked the C++ compiler to translate it to an executable or shared library for CPython to load. This was done even when embedding many modules into one (the "deep" compilation mode, option `--deep`).

This was simple to do and in theory ought to allow the compiler to do the most optimizations. But for large programs, the resulting source code could have exponential compile time behavior in the C++ compiler. At least for the GNU g++ this was the case, others probably as well. This is of course at the end a scalability issue of Nuitka, which now has been addressed.

So the major advancement of this release is to make the `--deep` option useful. But also there have been a performance improvements, which end up giving us another boost for the "PyStone" benchmark.

Bug fixes

- Imports of modules local to packages now work correctly, closing the small compatibility gap that was there.
- Modules with a "-" in the name are allowed in CPython. This lead to wrong C++ code created. (Thanks to Li Xuan Ji for reporting and submitting a patch to fix it.)
- There were warnings about wrong format used for `Ssize_t` type of CPython. (Again, thanks to Li Xuan Ji for reporting and submitting the patch to fix it.)
- When a wrong exception type is raised, the traceback should still be the one of the original one.
- Set and dict contractions (Python 2.7 features) declared local variables for global variables used. This went unnoticed, because list contractions don't generate code for local variables at all, as they cannot have such.
- Using the `type()` built-in to create a new class could attribute it to the wrong module, this is now corrected.

New Features

- Uses Scons to execute the actual C++ build, giving some immediate improvements.
- Now caches build results and Scons will only rebuild as needed.

- The direct use of `__import__()` with a constant module name as parameter is also followed in "deep" mode. With time, non-constants may still become predictable, right now it must be a real CPython constant string.

New Optimization

- Added optimization for the built-ins `ord()` and `chr()`, these require a module and built-in module lookup, then parameter parsing. Now these are really quick with Nuitka.
- Added optimization for the `type()` built-in with one parameter. As above, using from builtin module can be very slow. Now it is instantaneous.
- Added optimization for the `type()` built-in with three parameters. It's rarely used, but providing our own variant, allowed to fix the bug mentioned above.

Cleanups

- Using scons is a big cleanup for the way how C++ compiler related options are applied. It also makes it easier to re-build without Nuitka, e.g. if you were using Nuitka in your packages, you can easily build in the same way than Nuitka does.
- Static helpers source code has been moved to ".hpp" and ".cpp" files, instead of being in ".py" files. This makes C++ compiler messages more readable and allows us to use C++ mode in Emacs etc., making it easier to write things.
- Generated code for each module ends up in a separate file per module or package.
- Constants etc. go to their own file (although not named sensible yet, likely going to change too)
- Module variables are now created by the `CPythonModule` node only and are unique, this is to make optimizations of these feasible. This is a pre-step to module variable optimizations.

New Tests

- Added ExtremeClosure from my Python quiz, it was not covered by existing tests.
- Added test case for program that imports a module with a dash in its name.
- Added test case for main program that starts with a dash.
- Extended the built-in tests to cover `type()` as well.

Organizational

- There is now a new environment variable `NUITKA_SCONS` which should point to the directory with the `SingleExe.scons` file for Nuitka. The scons file could be named better, because it is actually one and the same who builds extension modules and executables.
- There is now a new environment variable `NUITKA_CPP` which should point to the directory with the C++ helper code of Nuitka.
- The "create-environment.sh" can now be sourced (if you are in the top level directory of Nuitka) or be used with eval. In either case it also reports what it does.
- To cleanup the many "Program.build" directories, there is now a "clean-up.sh" script for your use. Can be handy, but if you use git, you may prefer its clean command.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.4:

```
Pystone(1.1) time for 50000 passes = 0.34
This machine benchmarks at 147059 pystones/second
```

This is 91% for 0.3.4, up from 80% before.

Nuitka Release 0.3.3

This release of Nuitka continues the focus on performance. It also cleans up a few open topics. One is "doctests", these are now extracted from the CPython 2.6 test suite more completely. The other is that the CPython 2.7 test suite is now passed completely. There is some more work ahead though, to extract all of the "doctests" and to do that for both versions of the tests.

This means an even higher level of compatibility has been achieved, then there is performance improvements, and ever cleaner structure.

Bug fixes

Generators

- Generator functions tracked references to the common and the instance context independently, now the common context is not released before the instance contexts are.
- Generator functions didn't check the arguments to `throw()` the way they are in CPython, now they are.
- Generator functions didn't trace exceptions to "stderr" if they occurred while closing unfinished ones in "del".
- Generator functions used the slightly different wordings for some error messages.

Function Calls

- Extended call syntax with `**` allows that to use a mapping, and it is now checked if it really is a mapping and if the contents has string keys.
- Similarly, extended call syntax with `*` allows a sequence, it is now checked if it really is a sequence.
- Error message for duplicate keyword arguments or too little arguments now describe the duplicate parameter and the callable the same way CPython does.
- Now checks to the keyword argument list first before considering the parameter counts. This is slower in the error case, but more compatible with CPython.

Classes

- The "locals()" built-in when used in the class scope (not in a method) now is correctly writable and writes to it change the resulting class.
- Name mangling for private identifiers was not always done entirely correct.

Others

- Exceptions didn't always have the correct stack reported.

- The pickling of some tuples showed that "cPickle" can have non-reproducible results, using "pickle" to stream constants now

New Optimization

- Access to instance attributes has become faster by writing specific code for the case. This is done in JIT way, attempting at run time to optimize attribute access for instances.
- Assignments now often consider what's cheaper for the other side, instead of taking a reference to a global variable, just to have to release it.
- The function call code built argument tuples and dictionaries as constants, now that is true for every tuple usage.

Cleanups

- The static helper classes, and the prelude code needed have been moved to separate C++ files and are now accessed "#include". This makes the code inside C++ files as opposed to a Python string and therefore easier to read and or change.

New Features

- The generator functions and generator expressions have the attribute "gi_running" now. These indicate if they are currently running.

New Tests

- The script to extract the "doctests" from the CPython test suite has been rewritten entirely and works with more doctests now. Running these tests created increased the test coverage a lot.
- The Python 2.7 test suite has been added.

Organizational

- One can now run multiple "compare_with_cpython" instances in parallel, which enables background test runs.
- There is now a new environment variable "NUITKA_INCLUDE" which needs to point to the directory Nuitka's C++ includes live in. Of course the "create-environment.sh" script generates that for you easily.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.3:

```
Pystone(1.1) time for 50000 passes = 0.36
This machine benchmarks at 138889 pystones/second
```

This is 80% for 0.3.3, up from 66% before.

Nuitka Release 0.3.2

This release of Nuitka continues the focus on performance. But this release also revisits the topic of feature parity. Before, feature parity had been reached "only" with Python 2.6. This is of course a big thing, but you know there is always more, e.g. Python 2.7.

With the addition of set contractions and dict contractions in this very release, Nuitka is approaching Python support for 2.7, and then there are some bug fixes.

Bug fixes

- Calling a function with ** and using a non-dict for it was leading to wrong behavior. Now a mapping is good enough as input for the ** parameter and it's checked.
- Deeply nested packages "package.subpackage.module" were not found and gave a warning from Nuitka, with the consequence that they were not embedded in the executable. They now are.
- Some error messages for wrong parameters didn't match literally. For example "function got multiple..." as opposed to "function() got multiple..." and alike.
- Files that ended in line with a "#" but without a new line gave an error from "ast.parse". As a workaround, a new line is added to the end of the file if it's "missing".
- More correct exception locations for complex code lines. I noted that the current line indication should not only be restored when the call at hand failed, but in any case. Otherwise sometimes the exception stack would not be correct. It now is - more often. Right now, this has no systematic test.
- Re-raised exceptions didn't appear on the stack if caught inside the same function, these are now correct.
- For exec the globals needs to have "__builtins__" added, but the check was performed with the mapping interface. That is not how CPython does it, and so e.g. the mapping could use a default value for "__builtins__" which could lead to incorrect behavior. Clearly a corner case, but one that works fully compatible now.

Optimizations

- The local and shared local variable C++ classes have a flag "free_value" to indicate if an "PY_DECREF" needs to be done when releasing the object. But still the code used "Py_XDECREF" (which allows for "NULL" values to be ignored.) when the releasing of the object was done. Now the inconsistency of using "NULL" as "object" value with "free_value" set to true was removed.
- Tuple constants were copied before using them without a point. They are immutable anyway.

Cleanups

- Improved more of the indentation of the generated C++ which was not very good for contractions so far. Now it is. Also assignments should be better now.
- The generation of code for contractions was made more general and templates split into multiple parts. This enabled reuse of the code for list contractions in dictionary and set contractions.
- The with statement has its own template now and got cleaned up regarding indentation.

New Tests

- There is now a script to extract the "doctests" from the CPython test suite and it generates Python source code from them. This can be compiled with Nuitka and output compared to CPython. Without this, the doctest parts of the CPython test suite is mostly useless. Solving this improved test

coverage, leading to many small fixes. I will dedicate a later posting to the tool, maybe it is useful in other contexts as well.

- Reference count tests have been expanded to cover assignment to multiple assignment targets, and to attributes.
- The deep program test case, now also have a module in a sub-package to cover this case as well.

Organizational

- The gitweb interface might be considered an alternative to downloading the source if you want to provide a pointer, or want to take a quick glance at the source code. You can already download with git, follow the link below to the page explaining it.
- The "README.txt" has documented more of the differences and I consequently updated the Differences page. There is now a distinction between generally missing functionality and things that don't work in `--deep` mode, where Nuitka is supposed to create one executable.

  I will make it a priority to remove the (minor) issues of `--deep` mode in the next release, as this is only relatively little work, and not a good difference to have. We want these to be empty, right? But for the time being, I document the known differences there.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.2:

```
Pystone(1.1) time for 50000 passes = 0.39
This machine benchmarks at 128205 pystones/second
```

This is 66% for 0.3.2, slightly up from the 58% of 0.3.1 before. The optimizations done were somewhat fruitful, but as you can see, they were also more cleanups, not the big things.

Nuitka Release 0.3.1

This release of Nuitka continues the focus on performance and contains only cleanups and optimizations. Most go into the direction of more readable code, some aim at making the basic things faster, with good results as to performance as you can see below.

Optimizations

- Constants in conditions of conditional expressions (`a if cond else d`), `if`/`elif` or `while` are now evaluated to `true` or `false` directly. Before there would be temporary python object created from it which was then checked if it had a truth value.

  All of that is obviously overhead only. And it hurts the typically `while 1:` infinite loop case badly.

- Do not generate code to catch `BreakException` or `ContinueException` unless a `break` or `continue` statement being in a `try: finally:` block inside that loop actually require this.

  Even while uncaught exceptions are cheap, it is still an improvement worthwhile and it clearly improves the readability for the normal case.

- The compiler more aggressively prepares tuples, lists and dicts from the source code as constants if their contents is "immutable" instead of building at run time. An example of a "mutable" tuple would be `( {}, )` which is not safe to share, and therefore will still be built at run time.

  For dictionaries and lists, copies will be made, under the assumption that copying a dictionary will always be faster, than making it from scratch.

- The parameter parsing code was dynamically building the tuple of argument names to check if an argument name was allowed by checking the equivalent of `name in argument_names`. This was of course wasteful and now a pre-built constant is used for this, so it should be much faster to call functions with keyword arguments.

- There are new templates files and also actual templates now for the `while` and `for` loop code generation. And I started work on having a template for assignments.

Cleanups

- Do not generate code for the else of `while` and `for` loops if there is no such branch. This uncluttered the generated code somewhat.

- The indentation of the generated C++ was not very good and whitespace was often trailing, or e.g. a real tab was used instead of "t". Some things didn't play well together here.

  Now much of the generated C++ code is much more readable and whitespace cleaner. For optimizations to be done, the humans need to be able to read the generated code too. Mind you, the aim is not to produce usable C++, but on the other hand, it must be possible to understand it.

- To the same end of readability, the empty `else {}` branches are avoided for `if`, `while` and `for` loops. While the C++ compiler can be expected to remove these, they seriously cluttered up things.

- The constant management code in `Context` was largely simplified. Now the code is using the `Constant` class to find its way around the problem that dicts, sets, etc. are not hashable, or that `complex` is not being ordered; this was necessary to allow deeply nested constants, but it is also a simpler code now.

- The C++ code generated for functions now has two entry points, one for Python calls (arguments as a list and dictionary for parsing) and one where this has happened successfully. In the future this should allow for faster function calls avoiding the building of argument tuples and dictionaries all-together.

- For every function there was a "traceback adder" which was only used in the C++ exception handling before exit to CPython to add to the traceback object. This was now inlined, as it won't be shared ever.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.1:

```
Pystone(1.1) time for 50000 passes = 0.41
This machine benchmarks at 121951 pystones/second
```

This is 58% for 0.3.1, up from the 25% before. So it's getting somewhere. As always you will find its latest version here.

Nuitka Release 0.3.0

This release 0.3.0 is the first release to focus on performance. In the 0.2.x series Nuitka achieved feature parity with CPython 2.6 and that was very important, but now it is time to make it really useful.

Optimization has been one of the main points, although I was also a bit forward looking to Python 2.7 language constructs. This release is the first where I really started to measure things and removed the most important bottlenecks.

New Features

- Added option to control `--debug`. With this option the C++ debug information is present in the file, otherwise it is not. This will give much smaller ".so" and ".exe" files than before.
- Added option `--no-optimization` to disable all optimizations. It enables C++ asserts and compiles with less aggressive C++ compiler optimization, so it can be used for debugging purposes.
- Support for Python 2.7 set literals has been added.

Performance Enhancements

- Fast global variables: Reads of global variables were fast already. This was due to a trick that is now also used to check them and to do a much quicker update if they are already set.
- Fast `break`/`continue` statements: To make sure these statements execute the finally handlers if inside a try, these used C++ exceptions that were caught by `try`/`finally` in `while` or `for` loops.

  This was very slow and had very bad performance. Now it is checked if this is at all necessary and then it's only done for the rare case where a `break`/`continue` really is inside the tried block. Otherwise it is now translated to a C++ `break`/`continue` which the C++ compiler handles more efficiently.
- Added `unlikely()` compiler hints to all errors handling cases to allow the C++ compiler to generate more efficient branch code.
- The for loop code was using an exception handler to make sure the iterated value was released, using `PyObjectTemporary` for that instead now, which should lead to better generated code.
- Using constant dictionaries and copy from them instead of building them at run time even when contents was constant.

New Tests

- Merged some bits from the CPython 2.7 test suite that do not harm 2.6, but generally it's a lot due to some `unittest` module interface changes.
- Added CPython 2.7 tests `test_dictcomps.py` and `test_dictviews.py` which both pass when using Python 2.7.
- Added another benchmark extract from "PyStone" which uses a while loop with break.

Numbers

python 2.6:

```
Pystone(1.1) time for 50000 passes = 0.65
This machine benchmarks at 76923.1 pystones/second
```

Nuitka 0.3.0:

```
Pystone(1.1) time for 50000 passes = 0.52
This machine benchmarks at 96153.8 pystones/second
```

That's a 25% speedup now and a good start clearly. It's not yet in the range of where i want it to be, but there is always room for more. And the `break`/`continue` exception was an important performance regression fix.

Nuitka Release 0.2.4

This release 0.2.4 is likely the last 0.2.x release, as it's the one that achieved feature parity with CPython 2.6, which was the whole point of the release series, so time to celebrate. I have stayed away (mostly) from any optimizations, so as to not be premature.

From now on speed optimization is going to be the focus though. Because right now, frankly, there is not much of a point to use Nuitka yet, with only a minor run time speed gain in trade for a long compile time. But hopefully we can change that quickly now.

New Features

- The use of exec in a local function now adds local variables to scope it is in.
- The same applies to `from module_name import *` which is now compiled correctly and adds variables to the local variables.

Bug Fixes

- Raises `UnboundLocalError` when deleting a local variable with `del` twice.
- Raises `NameError` when deleting a global variable with `del` twice.
- Read of to uninitialized closure variables gave `NameError`, but `UnboundLocalError` is correct and raised now.

Cleanups

- There is now a dedicated pass over the node tree right before code generation starts, so that some analysis can be done as late as that. Currently this is used for determining which functions should have a dictionary of locals.
- Checking the exported symbols list, fixed all the cases where a `static` was missing. This reduces the "module.so" sizes.
- With gcc the "visibility=hidden" is used to avoid exporting the helper classes. Also reduces the "module.so" sizes, because classes cannot be made static otherwise.

New Tests

- Added "DoubleDeletions" to cover behaviour of `del`. It seems that this is not part of the CPython test suite.
- The "OverflowFunctions" (those with dynamic local variables) now has an interesting test, exec on a local scope, effectively adding a local variable while a closure variable is still accessible, and a module variable too. This is also not in the CPython test suite.
- Restored the parts of the CPython test suite that did local star imports or exec to provide new variables. Previously these have been removed.

- Also "test_with.py" which covers PEP 343 has been reactivated, the with statement works as expected.

Nuitka Release 0.2.3

This new release is marking a closing in on feature parity to CPython 2.6 which is an important mile stone. Once this is reached, a "Nuitka 0.3.x" series will strive for performance.

Bug Fixes

- Generator functions no longer leak references when started, but not finished.
- Yield can in fact be used as an expression and returns values that the generator user `send()` to it.

Reduced Differences / New Features

- Generator functions already worked quite fine, but now they have the `throw()`, `send()` and `close()` methods.
- Yield is now an expression as is ought to be, it returns values put in by `send()` on the generator user.
- Support for extended slices:

```
x = d[:42, ..., :24:, 24, 100]
d[:42, ..., :24:, 24, 100] = "Strange"
del d[:42, ..., :24:, 24, 100]
```

Tests Work

- The "test_contextlib" is now working perfectly due to the generator functions having a correct `throw()`. Added that test back, so context managers are now fully covered.
- Added a basic test for "overflow functions" has been added, these are the ones which have an unknown number of locals due to the use of language constructs `exec` or `from bla import *` on the function level. This one currently only highlights the failure to support it.
- Reverted removals of extended slice syntax from some parts of the CPython test suite.

Cleanups

- The compiled generator types are using the new C++0x type safe enums feature.
- Resolved a circular dependency between `TreeBuilding` and `TreeTransforming` modules.

Nuitka Release 0.2.2

This is some significant progress, a lot of important things were addressed.

Bug Fixes

- Scope analysis is now done during the tree building instead of sometimes during code generation, this fixed a few issues that didn't show up in tests previously.
- Reference leaks of generator expressions that were not fishing, but then deleted are not more.
- Inlining of exec is more correct now.

- More accurate exception lines when iterator creation executes compiled code, e.g. in a for loop
- The list of base classes of a class was evaluated in the context of the class, now it is done in the context of the containing scope.
- The first iterated of a generator expression was evaluated in its own context, now it is done in the context of the containing scope.

Reduced Differences

- With the enhanced scope analysis, `UnboundLocalError` is now correctly supported.
- Generator expressions (but not yet functions) have a `throw()`, `send()` and `close()` method.
- Exec can now write to local function namespace even if `None` is provided at run time.
- Relative imports inside packages are now correctly resolved at compile time when using `--deep`.

Cleanups

- The compiled function type got further enhanced and cleaned up.
- The compiled generator expression function type lead to a massive cleanup of the code for generator expressions.
- Cleaned up namespaces, was still using old names, or "Py*" which is reserved to core CPython.
- Overhaul of the code responsible for `eval` and `exec`, it has been split, and it pushed the detection defaults to the C++ compiler which means, we can do it at run time or compile time, depending on circumstances.
- Made `PyTemporaryObject` safer to use, disabling copy constructor it should be also a relief to the C++ compiler if it doesn't have to eliminate all its uses.
- The way delayed work is handled in `TreeBuilding` step has been changed to use closured functions, should be more readable.
- Some more code templates have been created, making the code generation more readable in some parts. More to come.

Optimizations / New Features

- Name lookups for `None`, `True` and `False` and now always detected as constants, eliminating many useless module variable lookups.
- As I start to consider announcing Nuitka, I moved the version logic so that the version can now be queries with `--version`
- More complete test of generator expressions.
- Added test program for packages with relative imports inside the package.
- The built-in `dir()` in a function was not having fully deterministic output list, now it does.

Summary

Overall, the amount of differences between CPython and Nuitka is heading towards zero. Also most of the improvements done in this release were very straightforward cleanups and not much work was required, mostly things are about cleanups and then it becomes easily right. The new type for the compiled generator expressions was simple to create, esp. as I could check what CPython does in its source code.

For optimization purposes, I decided that generator expressions and generator functions will be separate compiled types, as most of their behavior will not be shared. I believe optimizing generator expressions to

run well is an important enough goal to warrant that they have their own implementation. Now that this is done, I will repeat it with generator functions.

Generator functions already work quite fine, but like generator expressions did before this release, they can leak references if not finished , and they don't have the `throw()` method, which seems very important to the correct operation of `contextlib`. So I will introduce a decicated type for these too, possibly in the next release.

Nuitka Release 0.2.1

The march goes on, this is another minor release with a bunch of substantial improvements:

Bug Fixes

- Packages now also can be embedded with the `--deep` option too, before they could not be imported from the executable.
- Inlined exec with their own future statements leaked these to the surrounding code.

Reduced Differences

- The future print function import is now supported too.

Cleanups

- Independence of the compiled function type. When I started it was merely `PyCFunction` and then a copy of it patched at run time, using increasingly less code from CPython. Now it's nothing at all anymore.
- This lead to major cleanup of run time compiled function creation code, no more `methoddefs`, `PyCObject` holding context, etc.
- PyLint was used to find the more important style issues and potential bugs, also helping to identify some dead code.

Summary

The major difference now is the lack of a throw method for generator functions. I will try to address that in a 0.2.2 release if possible. The plan is that the 0.2.x series will complete these tasks, and 0.3 could aim at some basic optimizations finally.

Nuitka Release 0.2

Good day, this is a major step ahead, improvements everywhere.

Bug fixes

- Migrated the Python parser from the deprecated and problematic `compiler` module to the `ast` module which fixes the `d[a,] = b` parser problem. A pity it was not available at the time I started, but the migration was relatively painless now.
- I found and fixed wrong encoding of binary data into C++ literals. Now Nuitka uses C++0x raw strings, and these problems are gone.
- The decoding of constants was done with the `marshal` module, but that appears to not deeply care enough about unicode encoding it seems. Using `cPickle` now, which seems less efficient, but is more correct.

- Another difference is gone: The `continue` and `break` inside loops do no longer prevent the execution of finally blocks inside the loop.

Project Management

- I now maintain the "README.txt" in org-mode, and intend to use it as the issue tracker, but I am still a beginner at that.

  > ***Update***
  >
  > Turned out I never master it, and used ReStructured Text instead.

- There is a public git repository for you to track Nuitka releases. Make your changes and then `git pull --rebase`. If you encounter conflicts in things you consider useful, please submit the patches and a pull offer. When you make your clones of Nuitka public, use `nuitka-unofficial` or not the name `Nuitka` at all.
- There is a now a mailing list available too.

Reduced Differences

- Did you know you could write `lambda : (yield something)` and it gives you a lambda that creates a generator that produces that one value? Well, now Nuitka has support for lambda generator functions.
- The `from __future__ import division` statement works as expected now, leading to some newly passing CPython tests.
- Same for `from __future__ import unicode_literals` statement, these work as expected now, removing many differences in the CPython tests that use this already.

New Features

- The `Python` binary provided and `Nuitka.py` are now capable of accepting parameters for the program executed, in order to make it even more of a dropin replacement to `python`.
- Inlining of exec statements with constant expressions. These are now compiled at compile time, not at run time anymore. I observed that an increasing number of CPython tests use exec to do things in isolation or to avoid warnings, and many more these tests will now be more effective. I intend to do the same with eval expressions too, probably in a minor release.

Summary

So give it a whirl. I consider it to be substantially better than before, and the list of differences to CPython is getting small enough, plus there is already a fair bit of polish to it. Just watch out that it needs gcc-4.5 or higher now.

Nuitka Release 0.1.1

I just have just updated Nuitka to version 0.1.1 which is a bug fix release to 0.1, which corrects many of the small things:

- Updated the CPython test suite to 2.6.6rc, minimized much of existing differences.

- Compiles standalone executable that includes modules (with --deep option), but packages are not yet included successfully.
- Reference leaks with exceptions are no more.
- sys.exc_info() works now mostly as expected (it's not a stack of exceptions).
- More readable generated code, better organisation of C++ template code.
- Restored debug option `--g++-only`.

The biggest thing probably is the progress with exception tracebacks objects in exception handlers, which were not there before (always `None`). Having these in place will make it much more compatible. Also with manually raised exceptions and assertions, tracebacks will now be more correct to the line.

On a bad news, I discovered that the `compiler` module that I use to create the AST from Python source code, is not only deprecated, but also broken. I created the CPython bug about it, basically it cannot distinguish some code of the form `d[1,] = None` from `d[1] = None`. This will require a migration of the `ast` module, which should not be too challenging, but will take some time.

I am aiming at it for a 0.2 release. Generating wrong code (Nuitka sees `d[1] = None` in both cases) is a show blocker and needs a solution.

So, yeah. It's better, it's there, but still experimental. You will find its latest version here. Please try it out and let me know what you think in the comments section.

Nuitka Release 0.1 (Releasing Nuitka to the World)

Obviously this is very exciting step for me. I am releasing Nuitka today. Finally. For a long time I knew I would, but actually doing it, is a different beast. Reaching my goals for release turned out to be less far away than I hope, so instead of end of August, I can already release it now.

Currently it's not more than 4% faster than CPython. No surprise there, if all you did, is removing the bytecode interpretation so far. It's not impressive at all. It's not even a reason to use it. But it's also only a start. Clearly, once I get into optimizing the code generation of Nuitka, it will only get better, and then probably in sometimes dramatic steps. But I see this as a long term goal.

I want to have infrastructure in the code place, before doing lots of possible optimizations that just make Nuitka unmaintainable. And I will want to have a look at what others did so far in the domain of type inference and how to apply that for my project.

I look forward to the reactions about getting this far. The supported language volume is amazing, and I have a set of nice tricks used. For example the way generator functions are done is a clever hack.

Where to go from here? Well, I guess, I am going to judge it by the feedback I receive. I personally see "constant propagation" as a laudable first low hanging fruit, that could be solved.

Consider this readable code on the module level:

```
meters_per_nautical_mile = 1852

def convertMetersToNauticalMiles( meters ):
    return meters / meters_per_nautical_mile
def convertNauticalMilesToMeters( miles ):
    return miles * meters_per_nautical_mile
```

Now imagine you are using this very frequently in code. Quickly you determine that the following will be much faster:

```
def convertMetersToNauticalMiles( meters ):
    return meters / 1852
def convertNauticalMilesToMeters( miles ):
```

```
    return miles * 1852
```

Still good? Well, probably next step you are going to inline the function calls entirely. For optimization, you are making your code less readable. I do not all appreciate that. My first goal is there to make the more readable code perform as well or better as the less readable variant.