Constants

Constants

Python does not have Constants
...and how to ~~ab~~use them.

https://en.wikipedia.org/wiki/Constant_(computer_programming)
In computer programming, a constant is a value that cannot be altered by the program during normal execution

Ian Stewart
Hamilton Python User Group
20 April 2020

Constants. **What constants reside in Python?**

https://docs.python.org/3/library/constants.html

A small number of constants live in the built-in namespace. They are:

True
False
None
NotImplemented
Ellipsis
__debug__

What do constants created in Python code look like?

From PEP8: https://www.python.org/dev/peps/pep-0008/

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include:

```
BACKGROUND_COLOUR = "blue"
MAX_LIMIT = 100
VERSION = ("1", "9", "2")
```

Literal component may be any data type: String, Numeric, List , Tuple, etc.

Constants. **Variables in Linux that probably won't change**

Linux has many variables that may be accessed by a Python program. While a python program is running these variables may not change, so they could be considered to be constants. The "os" module provides:

```
>>> import os
>>> for key, value in os.environ.items():
...     print(key, value)
```

...returns about 50 items. For examples:

```
>>> os.environ["USER"]
'ian'

>>> os.environ["HOME"]
'/home/ian'

>>> os.environ["LANGUAGE"]
'en_NZ:en'
```

Constants. **More from the os module**

```
>>> os.sep
'/'

>>> os.path.expanduser("~")
'/home/ian'

>>> os.getuid()
1000  # Note: If its 0, then running with sudo priv.

>>> os.uname().sysname
'Linux'

>>> os.getcwd()
'/home/ian/development/constant-demo'

For the folder that the executing python program is in:
os.path.dirname(os.path.realpath(__file__)
...it maybe different from the getcwd() current working directory.
```

Constants. **The sys module**

```
>>> import sys

>>> sys.version_info
sys.version_info(major=3, minor=6, micro=9, releaselevel='final', ser
>>> sys.version_info.major
3
>>> sys.version
'3.6.9 (default, Nov  7 2019, 10:44:02) \n[GCC 8.3.0]'
>>> sys.platform
'linux'
>>> sys.ps1
'>>> '
>>> sys.ps2
'... '

print(sys.argv)
['constant_test.py']  # Program name is first argument on the list
```

Constants. **What's sysconfig module got for constants?**

```
>>> import sysconfig

>>> sysconfig.get_python_version()
'3.6'

>>> sysconfig.get_scheme_names()
('nt', 'nt_user', 'osx_framework_user', 'posix_home', 'posix_prefix',
'posix_user')

>>> sysconfig.get_path_names()
('stdlib', 'platstdlib', 'purelib', 'platlib', 'include', 'scripts',
'data')

>>> sysconfig.get_path('stdlib')
'/usr/lib/python3.6'

>>> sysconfig.get_path('scripts')
'/usr/bin'
```

Constants. **Any constants in the math module?**

```
>>> import math

>>> math.e
2.718281828459045

>>> math.pi  # π
3.141592653589793

>>> math.tau  # τ Number of radians in one turn. Python V3.6+
6.283185307179586
```

Constants. **OK, but...**

Where are the constants that show:

1. Where python will look for its modules

2. Where bash will look for files it can launch

Constants. **Where do python modules normally reside?**

```
>>> sys.path
['',
'/usr/lib/python36.zip',
'/usr/lib/python3.6',
'/usr/lib/python3.6/lib-dynload',
'/usr/lib/python3/dist-packages',
'/home/ian/.local/lib/python3.6/site-packages'
]
```

A **$ sudo pip3 install** of an application will place python modules here.

This ~/.local path was added after doing a **$ pip3 install** of an application.

Bash does not default to recognising these paths. In order for a program to be launched from the bash prompt some python code must reside in a bash $PATH and the python modules it calls reside in the above paths.

Constants. **Folders where executable files may be launched by bash**

```
>>> os.environ["PATH"].split(":")  or  >>> os.get_exec_path()
[
'/home/ian/.local/bin',
'/home/ian/bin',
'/usr/local/sbin',
'/usr/local/bin',
'/usr/sbin',
'/usr/bin',
'/sbin',
'/bin',
'/usr/games',
'/usr/local/games',
'/snap/bin',
]
```

These local paths were added to the $PATH once user “ian” created the folders.
They were added by ~/.profile.
Invoked after log out/in or $ source ~/.profile

From: ~/.profile

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
```

To add more paths: `sys.path.append('/path_to_python_scripts/')`

Constants. **Where do Python applications normally reside?**

Using **`$ sudo pip3 install`** ***`application`*** will “appear”[1] to place the program into: **`/usr/local/bin/`**<u>***`application`***</u>

Using **`$ pip3 install`** ***`application`*** will “appear”[1] to place the program into: **`/home/`**<u>***`USER`***</u>**`/.local/bin/`**<u>***`application`***</u>

In both cases it may be desirable to allow parameters to be changed to modify features of the program. E.g. If you don’t like the default of a red colour then: **`BACKGROUND_COLOUR = "blue"`**

This parameter modifying is normally provided by having a configuration file off the USER’s home directory:
`/home/`<u>***`USER`***</u>**`/.config/`**<u>***`application/application.conf`***</u>

[1] The reason for the using the word “appear” is explained in slides 26 & 27.

Constants. **Example: Installing an Internet Radio program**

System wide install:
`$ sudo pip3 install radio` will “appear” to place the program into **`/usr/local/bin/radio`**

Local install:
`$ pip3 install radio` will “appear” to place the program into **`/home/`*`USER`*`/.local/bin/radio`**

Downloaded from PyPI the program named “**radio**” starts with shebang of **#!/usr/bin/env python3.** The radio file is given execute permissions: **$ chmod +x radio**. This allows the program to be launched.

The configuration file is:
`/home/`*`USER`*`/.config/radio/radio.conf`

Constants. **Pip in virtual environments -- system and -- user**

System wide install:
`$ pip install --system radio` will “appear” to place the program into: **`/usr/local/bin/radio`**

Local install:
`$ pip install --user radio` will “appear” to place the program into: **`/home/`*`USER`*`/.local/bin/radio`**

`--user` Install to the Python user install directory for your platform. Typically **`~/.local/`** `on Linux`, or **`%APPDATA%\Python`** on Windows. On Debian systems, this is the default when running outside of a virtual environment and not as root.

`--system` Install using the system scheme (overrides **`--user`** on Debian systems)

Constants. **Code Guideline:**

The main part of a program should not include numeric or string data with the exceptions of 0, 1, True, False and None.

The data should be assigned to a constant or variable and this is used in the main program. The constant can be read by a class or function.

String in the main program:

```
class Main():
    def __init__(self):

        print("My_Program")

if __name__ == "__main__":

    Main()
```

Constant in the main program:

```
TITLE = "My_Program"

class Main():
    def __init__(self):

        print(TITLE)

if __name__ == "__main__":

    Main()
```

Constants. **Ian's rule:**

You can change the literal that is assigned to a constant as many times as you like in the first few milliseconds of launching your program.

After that you, once you enter the main section of your program, must treat it as a constant and not change it.

So why not just call them variables?

Because later on in your program you want UPPER CASE words in it that STAND OUT and you immediately know they are CONSTANTS.

OK, ...and please stop YELLING!

Constants. **Preceding entry to main program**

As the program launches it loads its built in constants

```
FONT_COLOUR = "red"

class Main():
    def __init__(self):

        print(FONT_COLOUR)

        # create gui window...
        widget.set_font_colour(FONT_COLOUR)

if __name__ == "__main__":

    Main()
```

Constant is defined

Console output:
red

The code would then be expected to create a GUI window and some widget would get to have its font colour set to the colour of the constant FONT_COLOUR

Constants. **Overriding the first value for the constant.**

As the program launches it loads its built in constants. This may be overridden further on in the program launch.

```
FONT_COLOUR = "red"

class Main():
    def __init__(self):

        print(FONT_COLOUR)

if __name__ == "__main__":

    FONT_COLOUR = "blue"
    Main()
```

Constant is defined

Console output:
`blue`

Override original constant

Constants. **Can not read the external constant**

Change the constant within the main program. Unable to read same named constant previously defined.

Constant is defined

```
FONT_COLOUR = "red"

class Main():
    def __init__(self):

        print(FONT_COLOUR)

        FONT_COLOUR = "green"

if __name__ == "__main__":

    FONT_COLOUR = "blue"
    Main()
```

Console output:

```
File "constant_test.py", line 6, in __init__
    print(FONT_COLOUR)
UnboundLocalError: local variable 'FONT_COLOUR'
referenced before assignment
```

Override constant again?

Override original constant

Constants

As the program launches it loads its built in constants, it then opens a configuration file and constants in this file over-ride built in constants.

```
FONT_COLOUR = "red"

class Main():
    def __init__(self):

        #print(FONT_COLOUR)

        FONT_COLOUR = "green"

        print(FONT_COLOUR)

if __name__ == "__main__":

    FONT_COLOUR = "blue"
    Main()
```

Constant is defined

Commented out

Constant local to the class Main() is defined

Console output:
green

Override original constant

Constants. **Override initial constant with a configuration file**

```
FONT_COLOUR = "red"

class Main():
    def __init__(self):

        print(FONT_COLOUR)

if __name__ == "__main__":
    with open("constant.conf", "r") as fin:
        configuration_list = fin.readlines()
    for line in configuration_list:
        line = line.strip()
        # Only excute if the line is a constant.
        if len(line) > 0 and line[0].isupper() and " = " in line:
            exec(line)
            print("Modified constant:", line)
    Main()
```

Constant is defined

Console output
yellow

constant.conf file:

```
#!/usr/bin/env python3
#
FONT_COLOUR = "yellow"
#
```

Console output
Modified constant: FONT_COLOUR = "yellow"

Constants. Use argparse to get --font-colour from command line

```
import argparse

FONT_COLOUR = "red"

class Main():
    def __init__(self):

    print(FONT_COLOUR)

if __name__ == "__main__":
    # Read config file code would go here.
    parser = argparse.ArgumentParser()
    parser.add_argument("-f", "--font-colour", type=str,
        dest='font_colour', default=FONT_COLOUR,
        help="Provide your desired font colour.")
    args = parser.parse_args()
    print(args.font_colour)
    FONT_COLOUR = args.font_colour

    Main()
```

Constant defined

Console output:
cyan

Console output:
cyan

Override original constant

Bash command:

```
$ python3 my_program.py --font-colour cyan
```

Constants. **Define once, overide twice...**

```
import argparse
FONT_COLOUR = "red"

class Main():
    def __init__(self):
        print(FONT_COLOUR)

if __name__ == "__main__"
    with open("constant.conf", "r") as fin:
        configuration_list = fin.readlines()
    for line in configuration_list:
        line = line.strip()
        if len(line) > 0 and line[0].isupper() and " = " in line:
            exec(line)

    parser = argparse.ArgumentParser()
    parser.add_argument("-f", "--font-colour", type=str, dest='font_colour',
                        default=FONT_COLOUR, help="Over-ride font colour.")
    args = parser.parse_args()
    FONT_COLOUR = args.font_colour
    Main()
```

1. red

Console output
cyan

2. yellow

3. cyan

```
$ python3 my_program.py --font-colour cyan
```

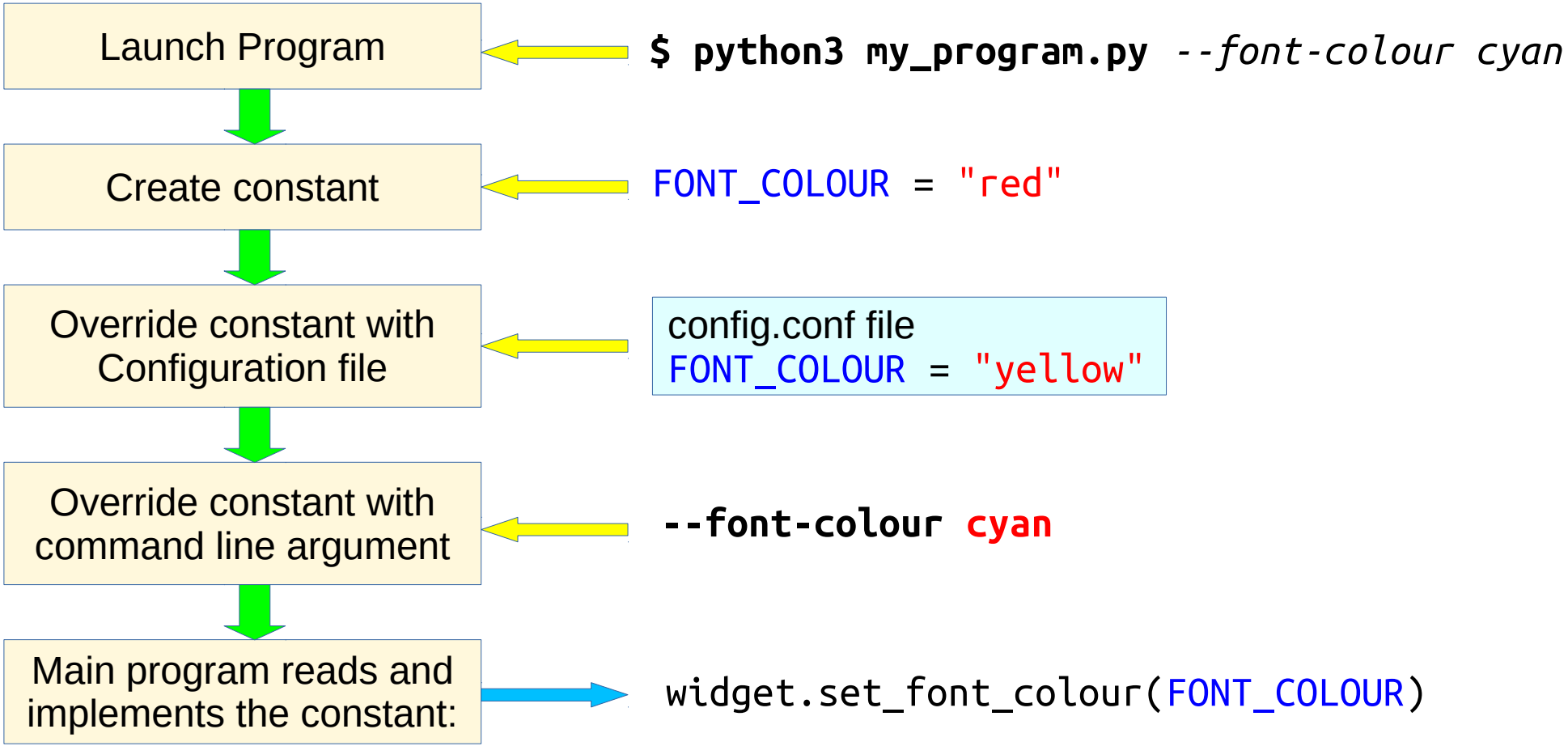
3. cyan

constant.conf file:

```
#!/usr/bin/env python3
#
FONT_COLOUR = "yellow"
```

2. yellow

Constants. **Summary of program launch**



Constants. **Example: Installing an Internet Radio program**

If **$ pip install radio** is used to install an internet radio program then we end up with:

`/home/`*`USER`*`/.local/bin/radio` is the script with execute permission in the bash $PATH that is used to launch the program with:
$ radio

`/home/`*`USER`*`/.local/lib/python3.6/site-packages/radio` is in the python path and has the files:
`radio.py` <--The main python program
`radio.conf` <-- The configuration file for the radio.py file

So what is the contents of the radio file that does the launching **`/home/`*`USER`*`/.local/bin/radio`** ?

Constants. **Example: Launching the Internet Radio program**

The bash command **$ radio** launches **/home/*USER*/.local/bin/radio** which is the following 8 lines of code:

Python searches its paths for radio.radio and finds:
/home/*USER*/.local/lib/python3.6/site-packages/radio/radio.py
This file radio.py has a “main” function.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import re
import sys
from radio.radio import main
if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw?|\.exe)?$', '', sys.argv[0])
    sys.exit(main())
```

Sort out python on Windows platform

The main function in the file radio.py is executed and the radio program is underway.

Constants. **Example: Launching the Internet Radio program**

If the program had been installed system-wide with **`$ sudo pip3 install radio`** the bash command **`$ radio`** would launch **`/usr/local/bin/radio`** with the same 8 lines of code:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import re
import sys
from radio.radio import main
if __name__ == '__main__':
    sys.argv[0] = re.sub(r'(-script\.pyw?|\.exe)?$', '',
sys.argv[0])
    sys.exit(main())
```

This time the python path where radio is found is:
`/usr/lib/python3/dist-packages/radio/radio.py`

Constants. **Example: Launching the Internet Radio program**

In the sudo install case the radio.conf file is installed in:
`/usr/lib/python3/dist-packages/radio/radio.conf`

In the local install case radio conf is installed in:
`/home/`*`USER`*`/.local/lib/python3.6/site-packages/radio/radio.conf`

In both cases, so the User can set their desired parameters it is preferable to have the radio.conf file in: **`~/.config/radio/radio.conf`**

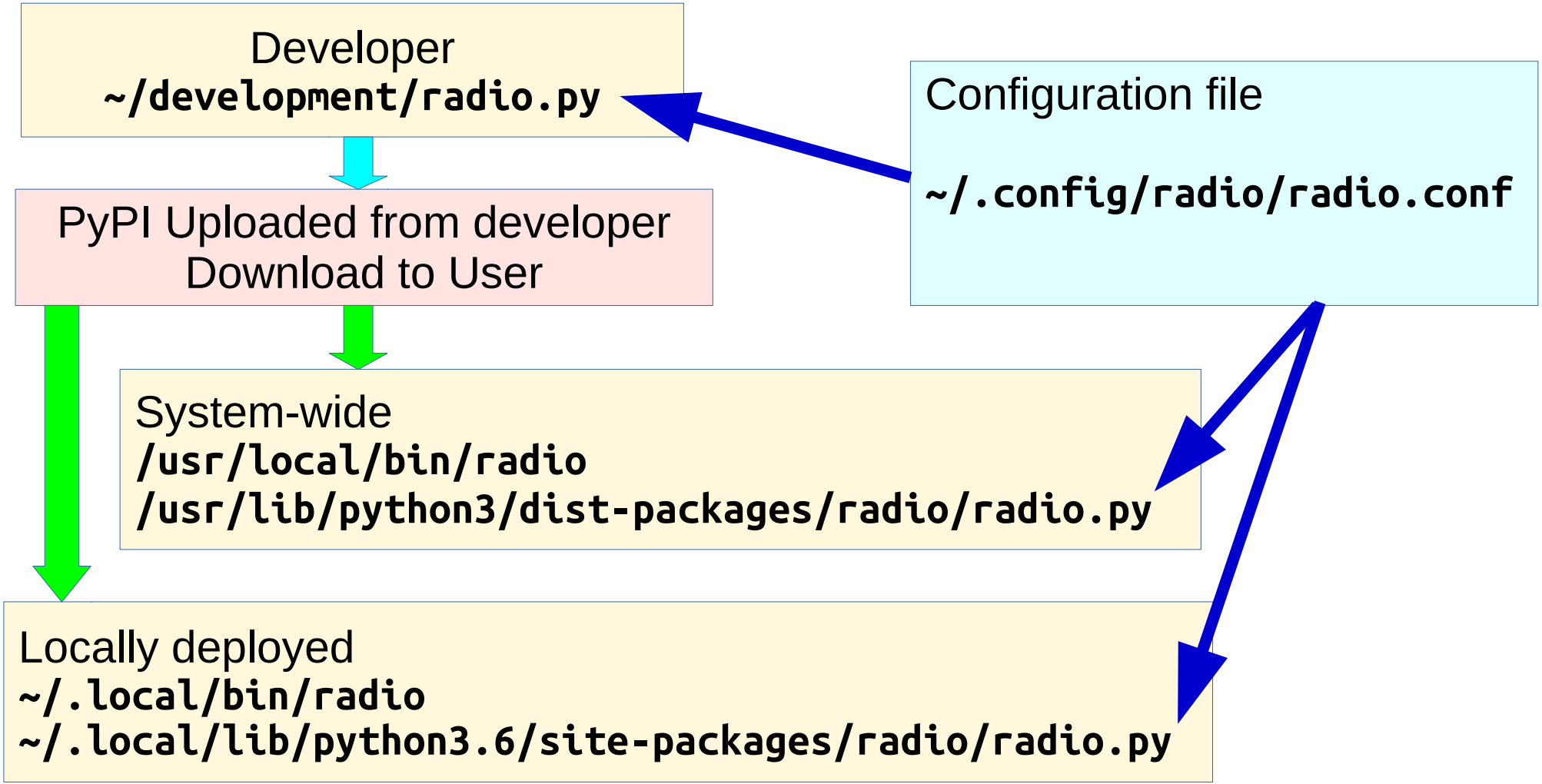
One method of doing this it to check on launching the radio program if the **`~/.config/radio/`** folder exists and if it has the file **`radio.conf`** in it. If not then make the directory if required and copy the radio.conf file into the directory.

On future launches, read from **`~/.config/radio/radio.conf`** and set the constants in the program to provide the desired User settings.

Constants. Code to include in program to move radio.conf.

```
import os
import shutil
# Get the path to where the distribution is located
path_dist = os.path.dirname(os.path.realpath(__file__))

# The home folder path is also required:
path_home = os.path.expanderuser("~")

# The ~/.config folder will exist. Need to created radio sub-folder:
try:
    os.mkdir(path_home + "/.config/radio")
except:
    pass

# The radio.conf file may now be copied to ~/.config/radio/radio.conf:
shutil.copy2(path_dist + "radio.conf",
             path_home + "/.config/radio/radio.conf")

# The User may now edit ~/.config/radio/radio.conf to set parameters.
```

Constants. **Configuration file must be available to each radio.py**



Constants.

Any Questions?

Launch your flame thrower now!

sys.exit(__end__)

Constants. **Appendix – Discussion of presentation**

Use of exec() function:

The proposed configuration file is really a python script. Although a test is performed for each line to attempt to ensure a constant is being modified, the use of exec() does provide a path for a malicious line of code to be executed.

Peter has previously delivered a presentation that may not have this security issue:
https://github.com/HamPUG/meetings/tree/master/2019/2019-11-11/console_scripts

I

Constants. Appendix – Discussion of presentation

Use of exec() function:

Ian has experimented with the following code, but not sure of the security risk:

```
# The folllowing loads a config file OK, but config file must end in .py
# The module name must not contain hyphens, but underscore is OK.
# https://stackoverflow.com/questions/67631/how-to-import-a-module-given-the-full

MODULE_PATH = "/home/ian/.config/radio-gui/radio_gui_conf.py"
MODULE_NAME = "radio_gui_conf"
import importlib
spec = importlib.util.spec_from_file_location(MODULE_NAME, MODULE_PATH)
module = importlib.util.module_from_spec(spec)
sys.modules[spec.name] = module
spec.loader.exec_module(module)

from radio_gui_conf import *
print(TITLE)
```

See: https://docs.python.org/3/library/importlib.html

Constants. Appendix – Discussion of presentation

Use of exec() function:

Possibly setuptools python module where its setup.cfg file might contain is a good way to get the files to ~/.config/ with a local install:

```
[options.data_files]
/etc/my_package =
    site.d/00_default.conf
    host.d/00_default.conf
```

The following needs to be tested:

```
[options.data_files]
~/.config/radio/ =
    radio.conf
```

See: https://setuptools.readthedocs.io/en/latest/setuptools.html#command-reference

Perhaps another option is to use configparser:

https://docs.python.org/3/library/configparser.html