Python for PHD-physicists
Part 2
(Nikhef 2018)

Taco Walstra

Overview

- Git branching and merging, stashing
- Python Debugger - Logging
- Combining Python and C (short + do-yourself)
- Object oriented programming
- Visual Python simulations in jupyter notebook

Git overview

- git add : “stage” files to be included in a commit
  git commit : do the commit to the local repository on your own system.
  (can be combined by using git commit -a -m”my new change whatever”
- git status : give the status of the files (modified files)
  git log : gives a list of the commits.
- We can move back to older version using:
  git checkout f25a34 somefile.py

Git overview(2)

- git push origin master : push the commits to a server
  git pull : pull the changes of others to your local copy.

  (We will cover the meaning of “origin and master” later)
- git clone http://urlwhatever/somerepo.git
  get a copy of a complete repository from a server.
- resolving a conflict between a server version and a local version:
  git pull results in a conflict.. Resolve it and push it version again.

Git - pull, fetch, merge

- Git pull consists of 2 components: fetch + merge
  - ☑ pull will change (merge) automatically the files and you will not see the what will be changed, unless a merge conflict occurs.
- In some cases it's preferred to do first a git fetch
  - ☑ Change your HEAD version to look like the FETCH_HEAD and execute git merge FETCH_HEAD afterwards

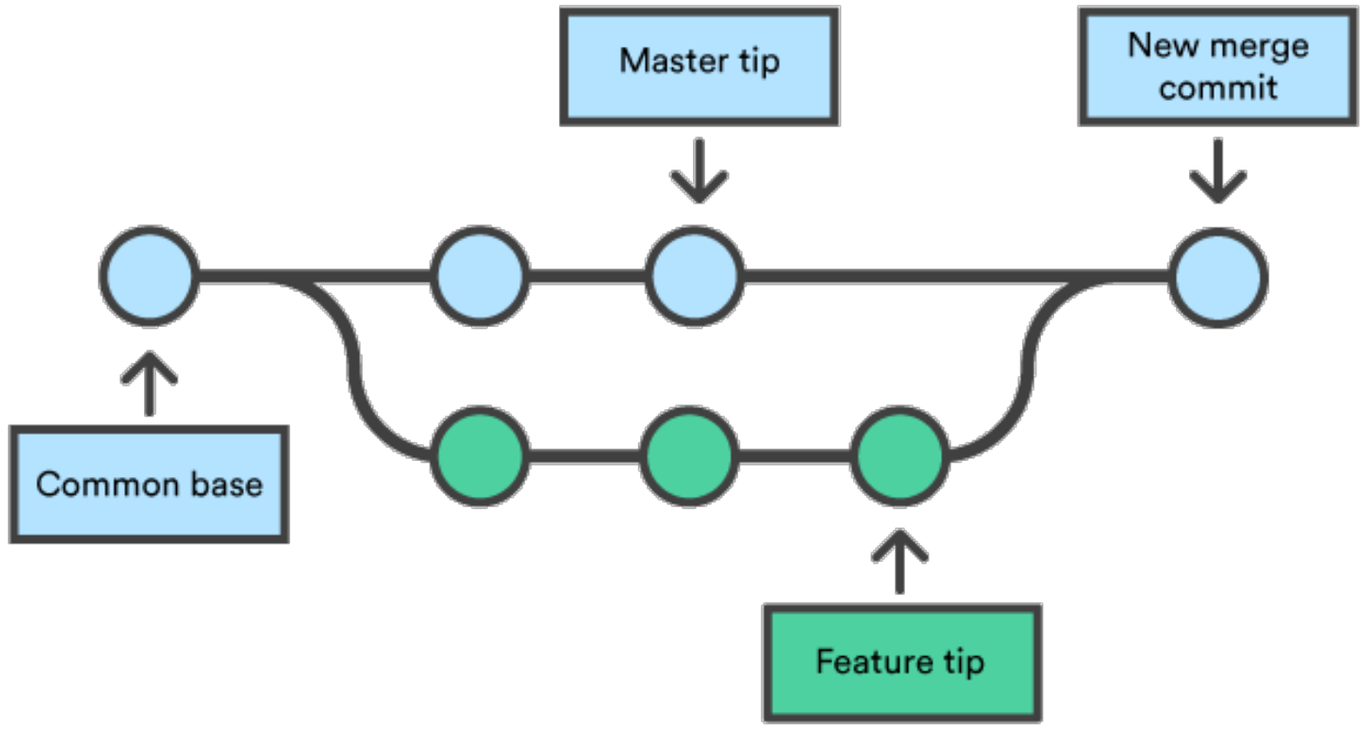
```
$git fetch
$git diff HEAD FETCH_HEAD
(this will display all changes which are to be merged)
diff --git a/another_version b/another_version
index be4fdef.c3b2f6 100644
--- a/another_version
+++ b/another_version
@@ -3,4 +3,4 @@ Tiny change
Small change 2
ABC DEF GHI JKL MNO PQR
ABC
-DEF
+ABC
```

git branch

- Sometimes you want to extend an application with some new functionality. It's recommended to do this in a branch, because
  - ☑ it allows to do a fast switching back to the root branch ('master')
  - ☑ it allows to work on both the new functionality and some bug fixing in the master branch

```
$git branch
* master
$git branch new_dev
$git branch
* master
  new_dev
```

```
$git checkout new_dev
$git branch
  master
* new_dev
```



git branch

- Creation:

```
git branch new_dev
git checkout new_dev
```

….do stuff

```
git commit -a -m”my new addition to new_dev branch”
git push origin new_dev
```

- Merging your branch back into the master:

```
git checkout master
git pull origin master
git merge my_branch
git push origin master
```

git push revisited

- git push origin master
  - it means: push the branch ‘master’ (i.e. the root branch) to the remote git server ‘origin’
  - Please neither names (origin and master) are fixed, but are mostly used in the git world. Keep to this standard.
  - origin is the remote which is added using the git command
    git remote add origin http://someurl
  - you can see the url using the command git remote -v

Git stash

- Stop temporarily your work and move to some old branch to fix something without committing incomplete work

```
$git status
# On branch master
nothing to commit, working directory clean
$git checkout our_new_branch
(now we starting working in this new branch, let's say on file mars.txt.
In the middle of the work we need to do someting on the
master branch. We try to move directly to the master branch:)
$git checkout master
error: Your local changes to the following files would be overwritten
 by checkout:
jupiter
Please, commit your changes or stash them before you can switch branches.
Aborting
(this explains all. We could commit the changes using git commit -a -m"my intermedia
commit" but we can also stash it temporarily:)
```

Git stash

```
$git stash
Saved working directory and index state WIP on
 another_fix_branch: 29c7e58 Renaming c and d.
HEAD is now at 29c7e58 Renaming c and d.
(Git stores the work in a commit, but it's reachable ONLY by the git stash command.
$git status
#On branch our_new_branch
nothing to commit, working directory clean
(As you see we moved to a clean situation of our branch, so we can also move to a di
$git checkout master
$git status
#On branch master
nothing to commit, working directory clean
(now to the changes to the master branch and move afterwards back to our_new_branch)
$git checkout our_new_branch
(You could have multiple stashes which can be listed:)
```

Git stash

```
$git stash list
stash@{0}: WIP on new_dev: bcdd64f change to jupiter
(this shows that I worked on the item jupiter. So now we reapply our previous change
our_new_branch:
$git stash pop
# On branch new_dev
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   saturn.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (47b320e2460550ac88f869623c17c5bd31aa8475)
(our stash list will be empty now)
```

Git

START USING IT!!!

pdb

- you can insert into your program:
  import pdb; pdb.set_trace()
  - ☑ will break into the debugger.
- see further https://docs.python.org/library/pdb.html
- demo bubblesort_dbg.py
-

pdb

- myscript.py:

```
……
def foo():
……

def main():
    do main stuff

if __name__==“__main__”:
     main()
```

- in idle:

```
import pdb
import myscript

pdb.run(‘myscript.main()’)
```

Logging

- import logging
  logger = logging.getLogger(‘myapplication’)
  logger.setLevel(logging.DEBUG)
  fh = logging.FileHandler(“error.log”)
  fh.setLevel(logging.DEBUG)

  logger.debug(“debug message”)
  logger.info("info message”)
  logger.warn(“warning!”)
  logger.error(“Error”)l
  logger.critical(“critical error”)
- Easy to handle and to remove!
- See also https://docs.python.org/howto/ogging-cookbook.html

Combining Python and C

- Create a Module in C following a strict “python” format
- Create a standard setup.py file which calls the module
- compile into a library (mymodule.so): python setup.py build
- Copy the .so library to your directory where you want to use the module or do a python setup.py install as root user (adding to /user/local/lib and site-packages.)
-

Classes - creating instances

```
class TemperatureControl(object):

      def getTemperature(self):
            temperature = acquireTemp()
            return temperature

t1 = TemperatureControl()
t1.getTemperature()
```

- t1 is a TemperatureControl object and has a function getTemperature
- last line is the same as: `TemperatureControl().getTemperature()`
- We can create multiple instances of the TemperatureControl class (suppose we have multiple temperature control instruments):

```
t1 = TemperatureControl()
t2 = TemperatureControl()
```

Classes - self

```
class TemperatureControl(object):

      def getTemperature(self):
            temperature = acquireTemp()
            return temperature

t1 = TemperatureControl()
t1.getTemperature()
```

```
type (TemperatureControl.getTemperature)
<class 'function'>
```

```
type (t1.getTemperature)
<class 'method'>
```

- In the second case we pass the object as the first argument to the corresponding function. **We call this parameter 'self'**.
- It's merely a convention; it's not a keyword!

Classes - self

```
class TemperatureControl(object):

        def getTemperature(self):
                temperature = self.acquireTemp()
                return temperature

        def acquireTemp(self):
                …….

t1 = TemperatureControl()
temp = t1.getTemperature()
```

- If you call a function from the same class reference the object:
  i.e. use again the self.functionname()
- What happens if you would call functionname()? (i.e. without “self”)

Classes - __init__

```
class TemperatureControl(object):
      def __init__(self, setpoint):
            self.setpoint = setpoint
            self. setInitialTemp()

      def getTemperature(self):
            temperature = self.acquireTemp()
            return temperature

      def setInitialTemp(self):
            #do some PID control with self.setpoint….

t1 = TemperatureControl(-273.0)
temp1 = t1.getTemperature()
```

- A class instantiation automatically looks up an __init__ method and executes it. Use this to initialise all your class variables.
- __init__(self, args) is NOT a “constructor” like in C++, java. Many sources are mistaken on this!

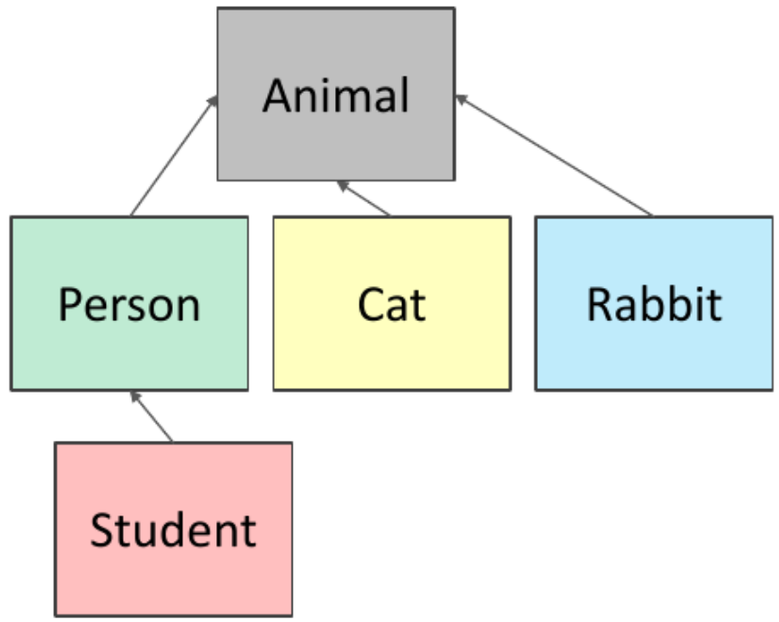
classes - private variables and functions

```
class TemperatureControl(object):
      __setpoint = 0 #with double underscore!!
      def __init__(self, setpoint):
            self.__setpoint = setpoint

      def getTemperature(self):
            temperature = self.acquireTemp()
            return temperature

      def   acquireTemp(self):
               …….

t1 = TemperatureControl(-273.0)
temp = t1.getTemperature()
t1.__setpoint = 4 => will not work
t1.__acquireTemp() => not allowed
```

classes - inheritance

- A class can be inherited from another class

```
class Cat (Animal):
    def speak(self):
        print("meow")
```



classes - inheritance

```
class Animal(object):
    def __init__ (self, name):
        print(name, “is an animal”)

class Cat(Animal):
     def __init__(self):
         print(“Cat says Meow’)
         super().__init__(‘Cat’)

c = Cat()
```

(source: animal_super.py)

Classes - multiple inheritance

```
class Animal(object):
    def __init__ (self, animalName):
        print(animalName, "is an animal")

class Mammal(Animal):
    def __init__(self,mammalName):
        print(mammalName, 'could be a mammal')
        super().__init__(mammalName)

class Bird(Animal):
    def   init  (self, birdName)
        print(birdName, 'could be a bird')
        super().__init__(birdName)

class Pet(Mammal, Bird):
    def __init__(self, petName):
        print(petName, 'is a pet')
        super(). __init__(petName)

c = Pet('Lorry')
```

#Lorry is a pet ; Lorry could be a mammal; Lorry could be a bird; Lorry is an animal

Example

- Some examples: class1-class3x
- class4: constructor with __new__
- class properties

Exercise

- Create a class “Vector” with methods to add to vectors, calculate the length extend to your own choice.
-

Exercise

- roman_number = ‘MMMCMLXXXVI’
  Convert this into an integer
- create a standard solution
- use a dictionary:
  roman_values = {‘I’:1, ‘V’: 5, ‘X’:10, ‘L’:50, ‘C’: 100, ‘D’:500, ‘M:’1000}
- Make a class of it.

- exercise 2: create a class UniversityPerson. It will store name, address and email of some University person.
  Create a subclass of the UniversityPerson for a student, which stores studentId and some other details. Use some functions to change details.

Visual Python simulations

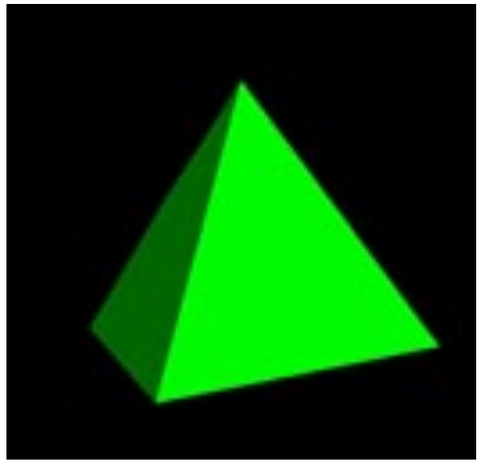
- Many examples especially the Physics, astronomy and chemistry field.
  - ☑ also in psychology (wrapped into psychopy) and life sciences
  - ☑ especially suited for graphical simulations
- Annoying aspect: visual package vs vpython package
  - ☑ Visual is the “old” library and not supported but many examples on internet have at the top: `from visual import *`
  - ☑ Use the vpython package and install it with conda: conda install vpython
  - ☑ Examples can be found on http://glowscript.org
- Warning: do not write a large framework without good internet search….

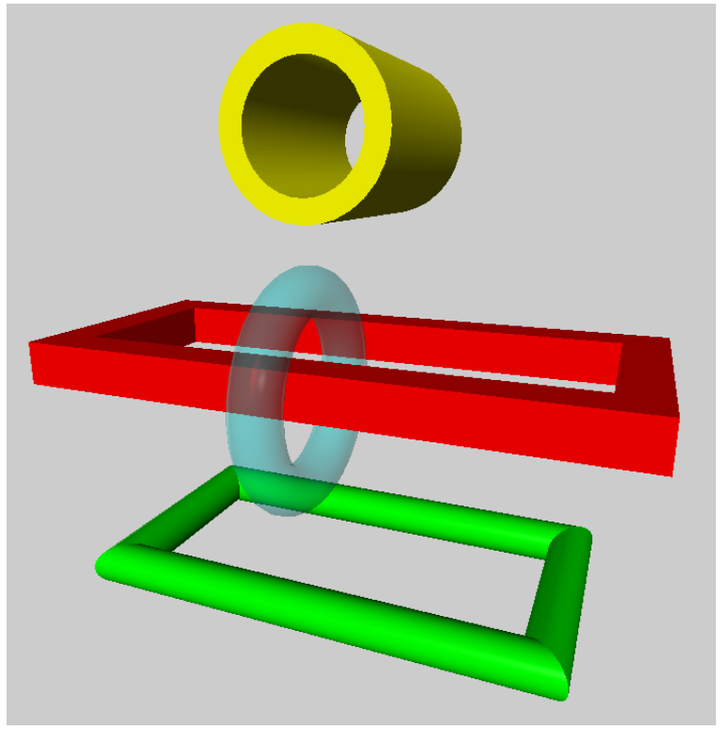
VPython - things you need to know

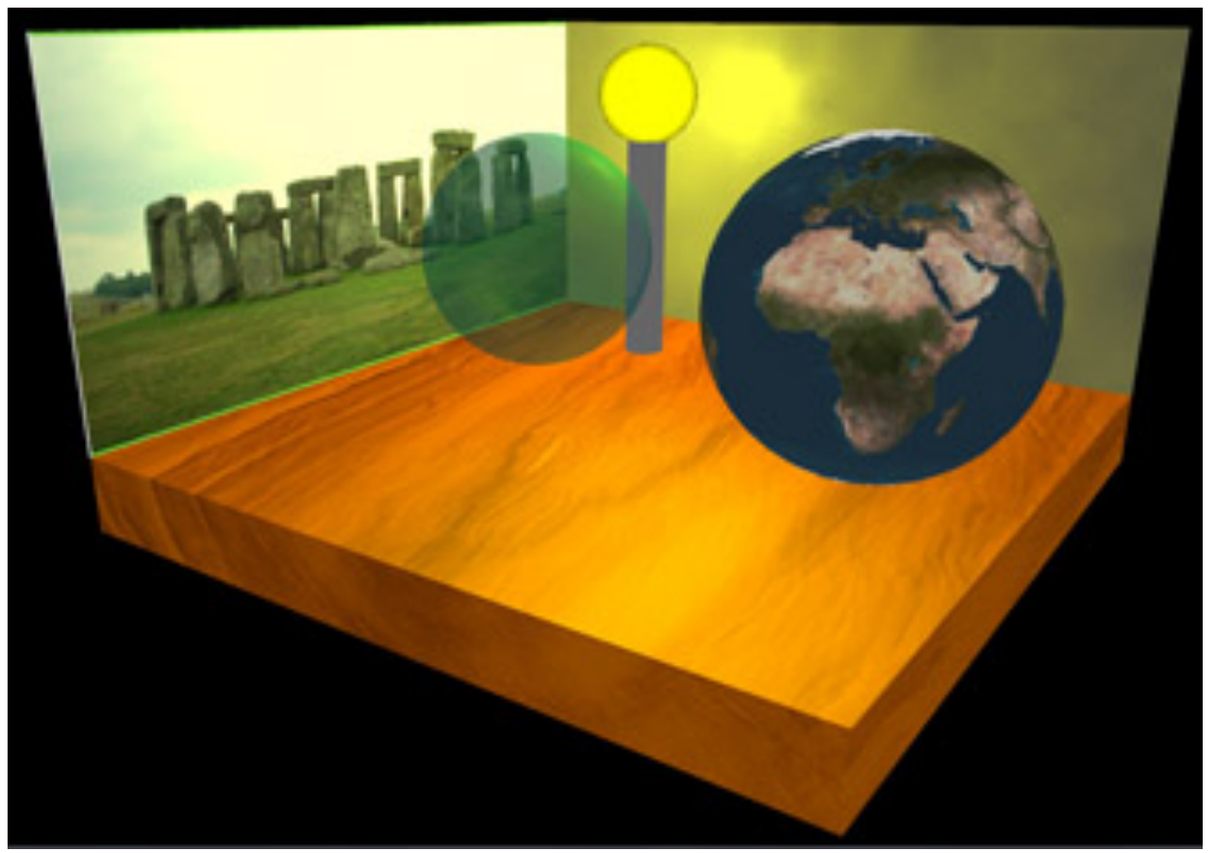
- Install vpython with conda (if not present): conda install vpython
- warning: the vpython package is newer than the “visual” package but unfortunately there are several backward incompatibilities
  - The older package only runs with a Python 2.7 installation and no longer receives updates or bug fixes
- vpython programs ONLY run in the browser (jupyter notebook)
- Try to use jupyter notebook with vphython environment to write your code!!
- See (some) examples at http://www.glowscript.org
-

vpython graphics

- Objects: sphere, box, cone, cylinder, ellipsoid, pyramid, ring, text,







Visual Python simulations

- git pull github.com/walstra/phd_nikhef
- in vpython several notebooks with the examples are available. A small exercise with a double star system as exercise (planets.pdf) is also in this directory.
- Remember: with increasing complexity of simulations a python class can be very useful.