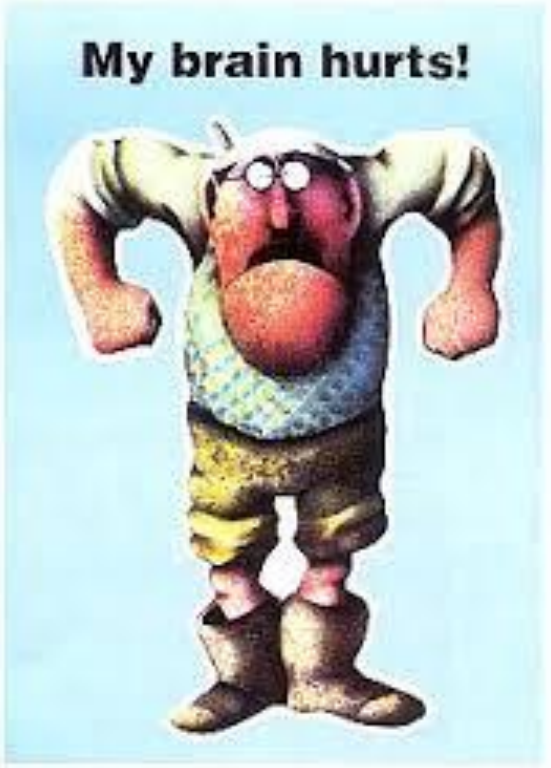
Programming in Python

Australian Synchrotron

May 2016

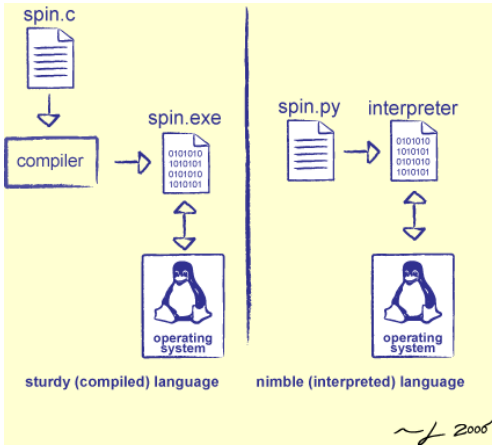
Python

- Free (cf. Matlab, IDL)
- Portable (Windows, Mac, Linux; write/run on all)
- Lots of extension modules
  - General - NumPy and SciPy
  - Field specific e.g. astro
- Popular across sciences
- Complements C/Fortran



Differences between C and Python

- Python code is highly readable
- Interpreted



- No variable declarations or function prototypes
- Time to solution:
  - Time to write a program (human time)
    - C is slow, Python is fast
  - Time for that program to run (machine time)
    - C is fast, Python is slow

Python versions

- Python distributions
  - CPython
    - python.org
    - Anaconda
    - Others: Python(x,y), Enthought Canopy
  - Jython, IronPython, PyPy
- 2.x versus 3.x
  - Use 3.x (3.5 at May 2016)

Python

- Interactive
  Default prompt >>>

```
>>> print("Hello world")
Hello world
>>>
```

- From a file

File hello.py

```
print("Hello world")
```

```
C:\>python hello.py
Hello world
```

- IPython shell

```
In [1]: print("Hello world")
Hello world
In [2]: "Hello world"
Out[2]: Hello world
```

Editors, Environments & IDEs

- Editors
  - Free, cross-platform: vim, EMACS, Atom
- IDEs
  - PyCharm
  - Spyder
- IPython shell
- Jupyter (Julia+Python+R)
  - IPython notebook
  - Web-based, interactive
  - Cluster support, good for communicating

Steps to programming (1 of 2)

- We need to represent and process data
- Represent the data
  - storage as files
    - need ways to read/write to/from working memory
  - program operation - variables and collections of data in memory
    - Numbers & Strings: e.g. Integer, Real (floating and fixed-point), Rational, Complex.
    - Collections: Lists, Tuples, Sets, Dictionaries, Arrays, Labelled Dataframes.

Steps to programming (2 of 2)

- Process the data
  - Program code
    - One or more statements to perform some task
    - Decision statements
    - Iteration (looping)

Variables (labels)

- Case sensitive
- In Python variables are created when something is assigned to them

```
>>> planet = "Pluto"
>>> moon = "Charon"
>>> p = planet
```

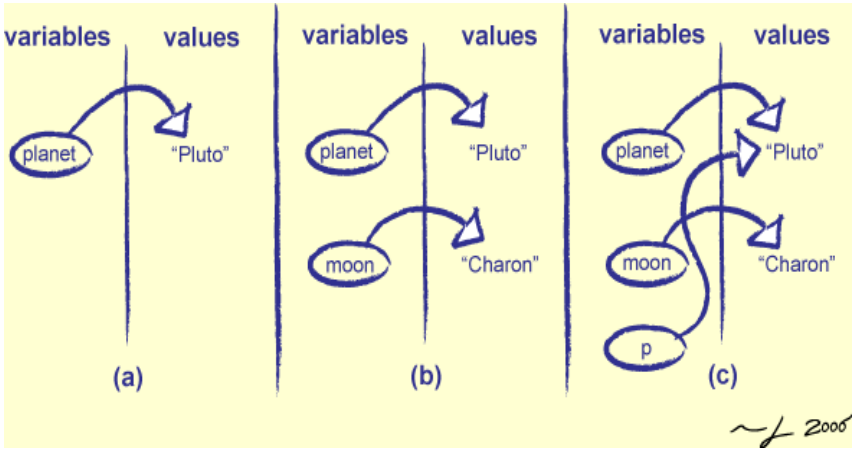


Figure: Variables Refer to Values

- A variable is just a name, and can refer to different types at different times

```
>>> planet = 9
```

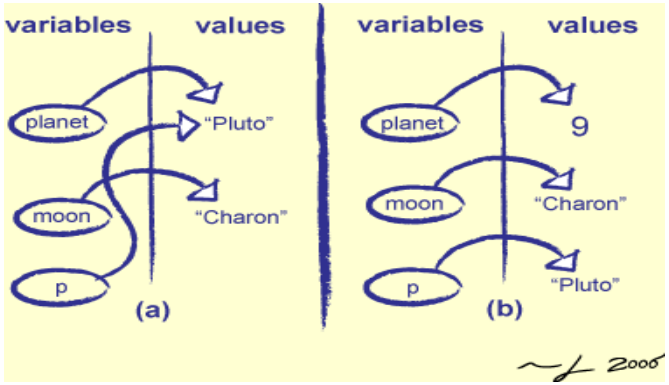


Figure: Variables Are Untyped

Types

int

```
-10  196

>>> 10**100          # a googol
10000000000000000000000000000000000
00000000000000000000000000000000000
0000000000000000000000000000000000
```

float

```
1.05   .6   2.   3e-5
```

- 64 bits (typically)
- ~17 sig. digits
- exponent -308→308

NoneType

```
None
```

str

```
'hello'  "hello"
"g'day"  'I say "hello"'
"""This sentence
goes over many lines"""
```

complex

```
1.5+0.5j

>>> a = 1 + 2j
>>> a.real # 1.0
>>> a.imag # 2.0
```

bool

- True `1`, False `0`

Operations & Typecasting

```
4 / 5                    # 0.8
  # 4 / 5 == 0 in Python 2.x;
  # from __future__ import division
6 // 4                   # 1 (div)
6 % 4                    # 2 (mod)
a,b = divmod(6,4)        # a=1,b=2
a / b                    # 0.5

int(0.6)                 # 0
int('100')               # 100
int('100', base=2)       # 4
int(0b100)               # 4
int(0x0f)                # 15
int(True)                # 1

str(100)                 # '100'
```

```
bool(1)          # True
bool(10)         # True
bool(0)          # False

ord('a')         # 97
chr(97)          # 'a'

(2+0j)/2         # 1+0j
complex(2,3)     # 2+3j

round(.7)        # 1.0
round(.777,2)    # 0.78
```

Object containers (collections)

- list (most common container)

```
fish = 'halibut'
[1, 2, 'three', ['four'], (3, 'x'), fish]
```

- tuple (constant *immutable* list)

```
(1, 2, 'three', ('four',))
```

- dictionary (aka hashtable)

```
{1:'one', 2:'dog'}
{'one':1, 'two':2}
{(0,1):11, (2,5):42}
```

- See also set type and collections module

Statements

- Statements are executed in order
- Indentation is used for grouping
  - Don't use tabs → convention 4 spaces
- Blank lines are ignored
- Comments #

```
print('hello')
print('goodbye')
```

Conditional execution

- Conditional test – **if** condition is **True** then do something **else** do something different

```
if colour == 'red':
    print('stop')
elif colour == 'green':
    print('go')
else:
    print('panic')
```

- condition is typically a comparison operation

  `<` `>` `<=` `>=` `==` `!=`

  or boolean logic operation (and, or, not)

```
if a == 0 and not (b > c or test(d)):
    print('hello')
```

Looping

- General looping – **while** condition is **True** do something

```
n = 0
while n < 2:
    n = n + 1
    print(n, end=' ')

while True:
    if n >= 4:
        break
    n = n + 1
    print(-n, end=' ')

# 1 2 -3 -4
```

Iterating

- **for** is used for iterating over
  - items in a list or array
  - characters in a string
  - lines in a file
  - types with an iterator defined
- Use **range()** to emulate C's **for** statement
  - upper bound is non-inclusive

```
for n in range(10):
    if n % 2:
        print(n, end=' ')                    # 1 3 5 7 9

for i in range(1, 11, 2):
    print(i, end=' ')                        # 1 3 5 7 9

for i, val in enumerate(['a', 'b']):
    print((i, val), end=' ')                 # (0, 'a') (1, 'b')
```

Strings

- Can concatenate, slice, search, split etc.
  - Concatenation

```
>>> 'baked ' + 'beans'          # 'baked beans'
```

  - Slicing

```
>>> s = 'beans spam'
>>> s[0]                        # 'b'
>>> s[1:3]                      # 'ea'
>>> s[:3]                       # 'bea'
>>> s[-4:]                      # 'spam'
```

  - Printing (cf. C sprintf)

```
>>> a = 10; b = 1.5; c = 1.5e-5
>>> 'spam{:03d}'.format(a)              # 'spam010'
>>> '{:3.2f} {:3.2g}'.format(b,c)       # '1.50 1.5e-05'
```

Lists (1 of 2)

- Contain any object type, including other lists
- Like an array but we will use an array module for true arrays
- Use it whenever you want to collect data together and access the data sequentially or randomly or sort it.
- Can use it like a stack or a queue
- Iterate (step through), sort and slice

Lists (2 of 2)

- Slicing cf. strings

```
things = ['spam', 'beans', 100, (1,2,3), ['1',2,3]]
print(things[0])      # 'spam'
del things[2:]        # ['spam', 'beans']
```

- All Python types have associated *methods* accessed via the dot "." operator

```
print(things)         # ['spam', 'beans']
things.append('eggs') # ['spam', 'beans', 'eggs']
things.reverse()      # ['eggs', 'beans', 'spam']
things.sort()         # ['beans', 'eggs', 'spam']
for i in things:
    print(i.upper())  # BEANS EGGS SPAM
```

Dot operator

- Refer to anything associated with the object
  - a field/variable: `baby.number_of_toes`
  - a method/function: `a_list.sort()`
  - a property: `complex_number.real`
  - a module: `numpy.fft`
- IPython and many editors support TAB-completion

Modules

- Python 3.5 about 200 built-in modules
- pypi.python.org about 78000 modules
- Access a module's functions and data like this:

```
import time                         # qualified access to time functions
print(time.localtime())             # (2006, 8, 28, 18, 8, 16, 1, 192, 0)

import numpy as np
a = np.array([[1,2],[2,3]])                    # create a 2x2 array

from scipy.special.basic import gamma          # gamma function
print(gamma(1.1))                              # 0.951350769867
```

Functions

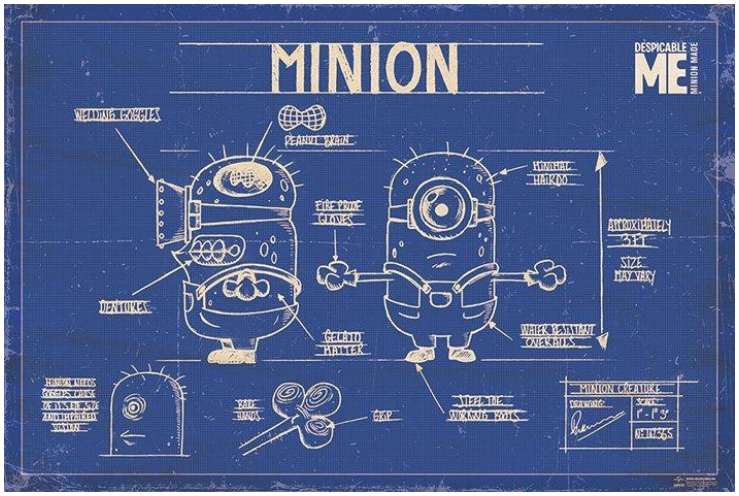
- A way of hiding information and reducing complexity
- Take arguments and return results

```
from numpy import exp

def rotate(x, y, angle):
    """Rotate a point about the origin by angle.

    Args:
        x (float): x-coordinate.
        y (float): y-coordinate.
        angle (float): angle in radian.

    Returns:
        (tuple): Tuple containing:
            (float, float): Rotated (x, y) coordinates.

    """
    z = (x + 1j*y) * exp(1j * angle)
    x1, y1 = z.real, z.imag
    return x1, y1

print(rotate(1, 0, pi))    # -1.0 1.22460635382e-016
print(rotate(1, 0, pi/2))  # 6.12303176911e-017 1.0
```

Classes

- Group data and associated functions together into 'classes'
  - data aka 'fields' and 'attributes'
  - member functions aka 'methods'

Class example



Bob Tim

```
class Minion():
    def __init__(self, name, number_of_eyes):
        self.name = name
        self.number_of_eyes = number_of_eyes
        self.number_bananas = 0

    def take_banana(self):
        self.number_bananas += 1

    def eat_all_bananas(self):
        self.number_bananas = 0

minion1 = Minion('Bob', number_of_eyes=1)
minion2 = Minion('Tim', number_of_eyes=2)
minion1.take_banana()
minion1.eat_all_bananas()
```

The NumPy module

- NumPy gives us multi-dimensional arrays containing C & Fortran-like datatypes.
  e.g. uint8, int64, float32, float64, float96, complex192, nan, inf, datetime
- cf. Matlab, IDL
- Reads and writes tabular data
- Use for series data, fields, images etc.
- Written in C so operations are *Fast*
- Elementwise and matrix-like functions
- numpy.random, numpy.fft, numpy.linalg

Scientific applications

- SciPy Stack: Python, NumPy, SciPy, Matplotlib, IPython, pandas, Sympy, nose
- Widely used for gluing and scripting legacy codes

Input/Output – Files (1 of 2)

- Read a file line by line, display it and write it to another file.

```
in_file = open("in.txt","r")    # open "in.txt" for reading
out_file = open("out.txt","w")  # create "out.txt" for writing

for line in in_file:
    print(line, end="")         # prevents print adding \n
    out_file.write(line)
```

- Open files are closed when the program ends, but you can do it using close()

```
in_file.close()
out_file.close()
```

Input/Output – Files (2 of 2)

- We don't need to read line-by-line. e.g. read a file into a string, display it and write it to another file.

```
with open("in.txt","r") as infile, \
     open("out.txt","w") as outfile:

    text = infile.read()
    print(text)
    outfile.write(text)
```

Exceptions and Tracebacks

- try/except example
- Traceback Example

```
try:
    important_value = 1/0
except ZeroDivisionError:
    pass
```

Some Useful modules

- PyTest
- NumPy, SciPy, Matplotlib, pandas
- PyEpics
- xraylib
- scikit-image, scipy.ndimage, OpenCV-Python
- imageio, h5py, csv
- Lmfit
- Flask
- requests, BeautifulSoup
- Click
- progressbar2
- astropy.units
- profilehooks

Resources

- URLs
  - confluence.synchrotron.org.au Search:Python
  - python.org
  - pythontutor.com
  - scipy.org
  - pyvideo.org
  - software-carpentry.org
- Learning more
  - Start here: python.org|Docs|Tutorial
  - codeacademy.com
  - Stack Overflow
  - Python cookbook

Fin