Introduction to Programming in Python

Kevin Rue-Albrecht
Oxford Biomedical Data Science Training Programme
2020-05-04 (updated: 2020-05-03)

Overview

UNIVERSITY OF OXFORD

- Python interpreters and text editors
- Variables
- Flow control - iteration and conditionals
- Functions
- Data structures
- Working with files
- Coding style
- Code organisation

The Python Programming Language

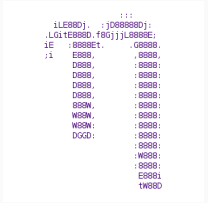
UNIVERSITY OF OXFORD

- Interpreted, high-level, general-purpose programming language (Van Rossum and Drake, 2009)
- Created by Guido van Rossum and first released in 1991
- Design philosophy: code readability, using significant whitespace
- Support for multiple programming paradigms
- Object-oriented, imperative, functional and procedural
- Large and comprehensive standard library

The Python Console

```
$ python
Python 3.7.6 | packaged by conda-forge | (default, Mar 23 2020, 23:03:20)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world!") # Python statement
Hello world!
>>>
```

Terminal Text Editors



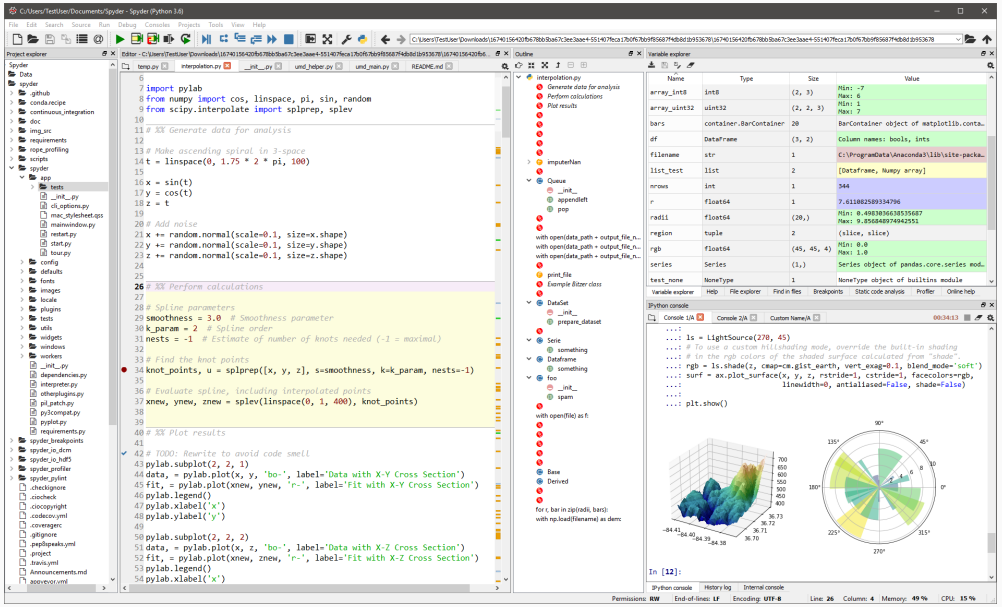
- No graphical interface required
- No mouse required
- Can be heavily customised
  - Language–specific syntax highlighting
  - Code auto-completion

Editors	Features
Vim, Emacs	[+] Been around for 40 years, very widely used, large number of add-ons. [-] Need to learn a set of keyboard shortcuts to use.
Nano	[+] Very simple and easy to use

Integrated Development Environments (IDE)

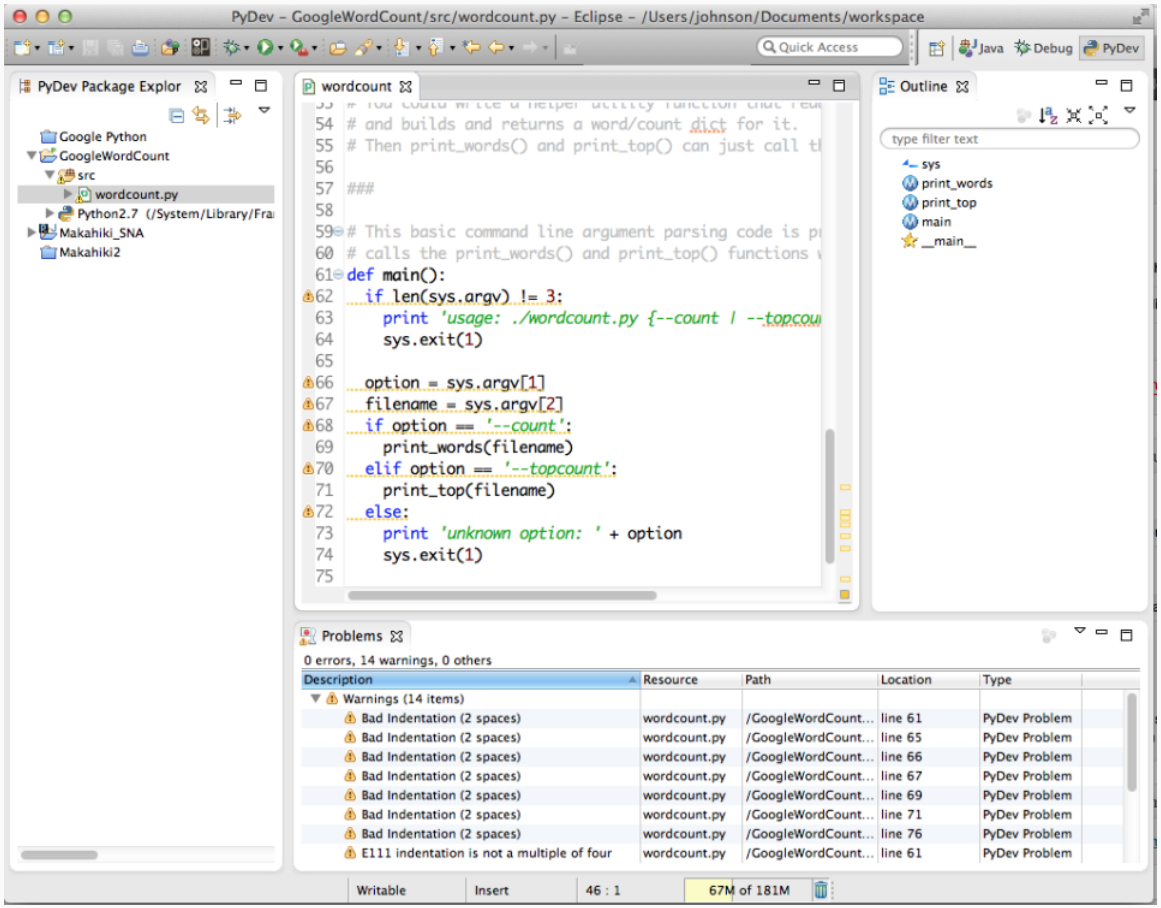
- For writing, testing, debugging and optimising reusable code
- Bring together programming tools into a single graphical interface
  - Text editor
  - Interpreter / console
  - Debugger
- Many different tools for Python

Spyder



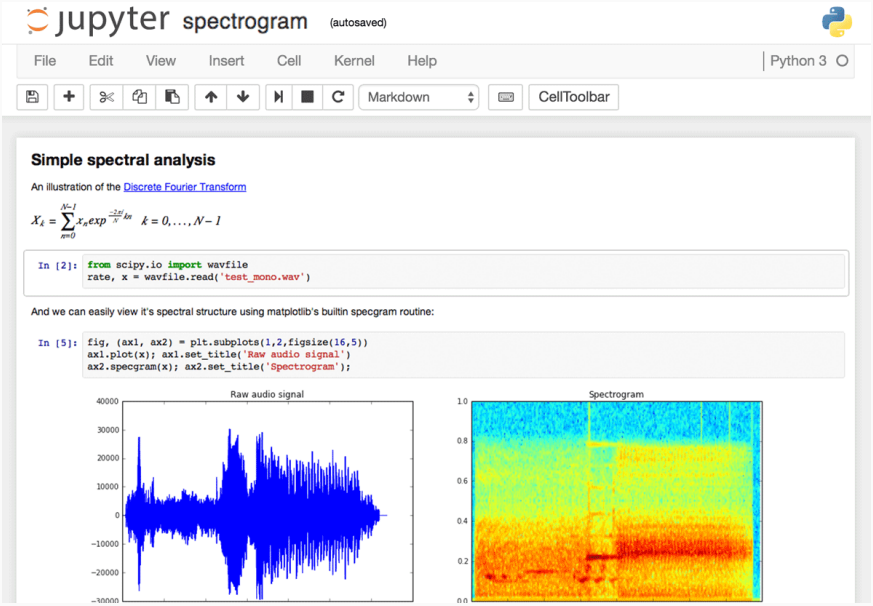
- iPython (interactive Python interpreter)
- Text editor
  - Syntax colouring
  - Code completion
  - Go to definition
- Debugging with pdb
- Code analysis
  - pyflakes
  - pylint
  - Syntax error checking
- Coding style – PEP8

PEP8



Extensive list of good and bad examples: https://www.python.org/dev/peps/pep-0008/

Jupyter Notebooks



- A lab book for exploratory data analysis
- Combines live code, equations, visualizations & narrative text
- Runs in your browser
- Build reports
- Shareable

- JupyterLab
  - Recently released
  - Replacing notebooks

Variables in Python

UNIVERSITY OF OXFORD

- A variable is a named memory location to store piece of data
- Contents can vary
- Variables have a type (e.g., number, string)
- Names must start with a letter or underscore
- Can contain only letters, numbers and underscore
- Case sensitive

Variables Types

- Numbers
  - Integer
  - Float
- Strings
  - Sequence of characters
  - Can use single or double quotes
  - Multiline strings use triple quotes
    - `"""`
    - `'''`
- Boolean
  - `True` or `False`

Assignment using `=` operator

```
a = 5
b = 0.01
c = 6e12
```

```
course = "obds"
talk = 'Python'
description = '''Introduction
to programming in Python'''
```

```
new_script = True
```

Operators

- Assignment
  - `x = 5`
- Arithmetic
  - `+`, `-`, `*`, `/`, `%`
  - Shortcuts `+=`, `-=`, `*=`
- Comparison
  - `>`, `<`, `>=`, `<=`, `==`, `!=`
- Logical
  - `and`, `or`, `not`

```
x = x + 1
x += 1
```

```
x > 1
```

```
x > 1 and x < 3:
```

Flow Control - Conditionals

- Make a decision
- Logical test
- Execute different code depending on outcome
- `if` statement

```
number = 23
guess = 12

if guess == number:
    # New block starts here
    print('Congratulations, you guessed it.')
    print('(but you do not win any prizes!)')
    # New block ends here
elif guess < number:
    # Another block
    print('No, it is higher than that')
else:
    print('No, it is lower than that')
    # you must have guessed > number to reach here
```

Flow Control - Iteration

The `for` loop

- Repeat code a specified number of times

The `while` loop

- Repeat while a condition is true

Flow control

- `break` – stop looping
- `continue` – skip to the next iteration of the loop

```
running = True
while running:
    guess = int(input('Enter an integer : '))
    if guess == number:
        print('Congratulations!')
        # this causes the while loop to stop
        running = False
    elif guess < number:
        print('No, it is higher than that.')
    else:
        print('No, it is lower than that.')
```

Functions

- Give a name to a block of statements
  - A group of instructions with a specific goal
- Reuse the code in multiple places
- Defined using the `def` keyword

```
def say_hello():
    # block belonging to the function
    print('hello world')

say_hello()  # call the function
```

Why Functions?

- Divide and conquer
  - Divide complicated tasks into simpler and more manageable ones
- Avoid writing redundant code
  - Functions can be reused either in the same or in different scripts
- Enhance the readability of the code
- Easier testing and debugging
  - Errors can be easily located

Passing data to a function

- Parameters – values you supply to a function
- Can have named parameters (keyword arguments)
- Parameters can have default values

```
def print_max(a, b):
    if a > b:
        print(a, 'is maximum')
    elif a == b:
        print(a, 'is equal to', b)
    else:
        print(b, 'is maximum')

# call the function
print_max(3, 4)
```

```
def func(a, b=5, c=10):
    print('a=', a, ', b=', b, '& c=', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

Named parameters can be given in any order.

Returning data from a function

- The `return` statement is used to return (i.e., exit) from a function.
- Can optionally return a value from the function as well.

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'The numbers are equal'
    else:
        return y

print(maximum(2, 3))
```

Example: Reverse complement of DNA

```
def complement_base(base):
    output = None
    if base == 'A':
        output = 'T'
    elif base == 'G':
        output = 'C'
    elif base == 'C':
        output = 'G'
    elif base == 'T':
        output = 'A'
    else:
        print("Unknown base")
    return output
```

A function can be reused in many places.

```
sequence = "AAAACCCGGT"

complement = []

for base in sequence:
    complement.insert(0, complement_base(base))
```

Calling a function is much simpler.

Variable Scope

- Variables declared inside a function are not accessible outside
  - *Local* variables
- All variable are accessible only within the block where they are declared
  - Variable *scope*
- This can be overcome by assigning *global* variables
  - Assign in outermost code block

Data structures

- Used to store a collection of related data
  - List - ordered collection of items
  - Tuple – similar to lists but immutable (cannot be changed)
  - Dictionary – key-value pairs
- Sequences
  - Lists, tuples and strings are all sequences
  - Features:
    - Membership tests (`in`, `not in`)
    - Indexing operations (retrieve by position)

Lists

- Ordered collection of items
- Add and remove items
- Sort the list
- Iterable (can easily loop over them)
- Denoted using square brackets `[]`

```
shoplist = ['apple', 'mango', 'carrot']
items = len(shoplist)
print('I have ', items, ' items to buy')

print('These items are: ')
for item in shoplist:
    print(item, end=' ')

# add item to list
shoplist.append('rice')
print('My list is now', shoplist)

# Sort the list
shoplist.sort()
print('Sorted list is', shoplist)

# Access list items by position (0-based index)
print('The first item is', shoplist[0])

# Delete item from list
olditem = shoplist.pop(0)
print('I bought the', olditem)
print('My shopping list is now', shoplist)
```

Slicing lists

- Take a subset of items from a list
- Square bracket notation
- 0-based counting

```
my_list = ['p','r','o','g','r','a','m']
# from 3rd to 5th
print(my_list[2:5])

# from beginning until 4th from end
print(my_list[:-3])

# from 1st to 3rd
print(my_list[:3])

# from 6th to end
print(my_list[5:])

# from beginning to end
print(my_list[:])
```

Nesting lists

- Elements within data structures can be data structures themselves
- Allows creation of high dimensional structures

```
              str    float      int       list
nested = ["hello",     2.0,       5,  [10, 20]]
nested[3][1]
20
```

Syntax

`nested[outer_index][inner_index]`

Copying Lists

- Lists can be very large
- You don't want to copy them by default
- Data structures are *objects*
- When you assign an object to a variable
  - The variable contains a memory address for the object
  - Not the object itself

```
shoplist = ['apple', 'mango', 'carrot']
mylist = shoplist
# mylist is another name pointing to same object!

# Remove item from list
del shoplist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# They are both the same - point to same object

# Make a copy by doing a full slice
mylist = shoplist[:]
# Remove first item
del mylist[0]

print('shoplist is', shoplist)
print('mylist is', mylist)
# now they are different
```

Tuples

- Immutable list
- Specified using round brackets `()`
- Slicing is the same as for lists, using square bracket notation `[]`

```
zoo = ('python', 'elephant', 'penguin')
print('Number of animals in the zoo is', len(zoo))

new_zoo = ('monkey', 'camel', zoo)
print('Number of cages in the new zoo is', len(new_zoo))
print('All animals in new zoo are', new_zoo)
print('Animals brought from old zoo are', new_zoo[2])
print('Last animal brought from old zoo is', new_zoo[2][2])
print('Number of animals in the new zoo is', len(new_zoo)-1+len(new_zoo[2]))
```

Dictionaries

- Like an address-book
- Associate keys with values
- Key must be unique
- Keys immutable
  - String, int, tuple
- Specified using curly brackets
- Not ordered
  - Randomised for security reasons

```
ab = {
  'David': 'david.sims@imm.ox.ac.uk',
  'Charlie': 'charlotte.george@imm.ox.ac.uk',
  'Spammer': 'spammer@hotmail.com'
}
print("David's email is", ab['David'])

print('There are ', len(ab), ' contacts')

# Returning a value
email = ab.get('David', 0)
# Returns 0 if not found

# Deleting a key-value pair
del ab['Spammer']

# Adding a key-value pair
ab['Guido'] = 'guido@python.org'

# Searching the keys
if 'Guido' in ab:
    print("Guido's address is", ab['Guido'])
```

Iterating over dictionaries

```python
for name in ab.keys():
    print(name)

for address in ab.values():
    print(address)

for name, address in ab.items():
    print(f'Mail {name} at {address}')
```

F-string for pretty printing.

Copying dictionaries

- Need to copy explicitly (like lists)

```
ab2 = dict(ab)
ab2 = ab.copy()
```

Make a shallow copy (does not copy nested data structures)

```
import copy
ab2 = copy.deepcopy(ab)
```

Make a deep copy (copies nested data structures)

Working with files

- Python can be used to read and write files
- Can also read and write compressed files (gzip package)

```
# Open for 'w'riting
f = open('seq.fa', 'w')
# Write text to file
f.write(sequence)
# Close the file
f.close()
```

Can open for read, write or append.

```
# Open for 'r'eading
with open('seq.fa', 'r') as f:
    # iterate line by line
    for line in f:
        print(line)
# Don't need to close the file
```

When you use `with` you don't need to close the file File closing handled even when code crashes.

Best practice

String Manipulation

- A string is an object and a sequence
- There are many methods for manipulating strings
- Strings are sliceable

```
name = 'WIMM'

if name.startswith('WI'):
    print('name starts with "WI"')

if 'M' in name:
    print('Name contains the letter "M"')

if name.find('IMM') != -1:
    print('Name contains the string "IMM"')

# Slice from beginning to 2nd characters
print(name[:-2])

# Slice from 3rd to last characters
print(name[2:])
```

Credits: https://www.programiz.com/python-programming/methods/string

Writing Python Code – Style Guide

- One statement per line
- Indentation (enforced)
  - Indent each code block for readability
- Variable and function naming conventions (PEP8)
  - All lowercase
  - Use underscores to separate words
- Style guides – (More PEP8)
  - Spaces not tabs for indentation
  - Line length maximum 79 characters
  - Blank lines around functions and classes
  - Import statements at top, one per line

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

Code Organisation

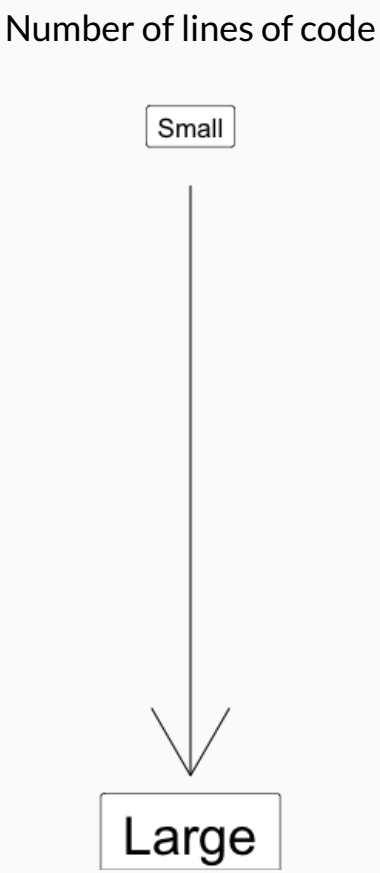
Organisation levels

- Functions
- Scripts
- Modules
- Packages
- Libraries

Code Organisation

Organisation levels

- Functions
- Scripts
- Modules
- Packages
- Libraries



Code Organisation

Rule of thumb

Type	Size
Function	Fits your screen.
Script	100's lines. Average CGAT script: 320 lines.
Module	100's lines. Average CGAT script: 630 lines.
Package	100k's lines. `pandas` has 1M lines of code.
Library	The Python standard library has 1M lines of code.

What is a script?

- A file with instructions to be executed in a particular order
- The execution begins in a special function called the `main` function
- Can be run on the command line
- Can specify parameters on the command line (optional)

How to organise a script?

```
#!/usr/bin/env python # needed to run on the command line

"""
Docstring
- Tells the use what the script does
- Used for building documentation
"""
# Import useful code from existing packages ---
import sys
import math
# Global variables ---------------------------
variable1 = 1
variable2 = "test"
# Functions ----------------------------------
def do_this():
    # Code block
# Main function ------------------------------
def main():
    do_this()
# Trigger main function (command line) --------
if __name__ == "__main__":
    main()
```

Modules

UNIVERSITY OF OXFORD

What is a module?

- Python file containing a set of functions with a common theme
- Unlike scripts, modules are meant to be imported
- Do not contain a `main` function

Example

```
import os
os.getcwd()
```

Software Packages

UNIVERSITY OF OXFORD

What is a package?

- A folder with a set of files (and/or subfolders)
    - Code
    - Documentation
    - Tests
    - etc.

e.g. pandas, matplotlib

Software Libraries

What is a library?

- A collection of packages
- It normally exposes an **API**
  - Application Programming Interface
- Used to develop new software on top of it

e.g. tensorflow: Google's software library for Machine Learning

Python Standard Library

What is it?

- Collection of packages: sys, math, csv, ...
- Each package contains modules
- Each modules contains functions
- So you can build *scripts*!

Reference: https://docs.python.org/3/library/index.html

Goal

By the end of the week:

- Functions (write)
- Scripts (write)
- Modules (use)
- Packages (use)
- Libraries (use)

Number of lines of code

Credits

- David Sims for sharing his original slides.

Further reading

- Python For Beginners - Getting Started
- Python Basic - Exercises, Practice, Solutions

References

Van Rossum, G. and F. L. Drake (2009). *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace. ISBN: 1441412697.