Python Programming 101

Gianluca Campanella

Today's class

Prerequisites

- No previous programming experience required
- Lots of slides with `actual code`

Goals

- Make you self-sufficient as a Python beginner
- Teach you a little bit about Python's culture

Why Python?

Ideal for scripting and rapid prototyping

- General-purpose, high-level language
- Elegant syntax
- Interpreted
- ‘Comes with batteries’ (lots of them!)
- The Zen of Python: `import this`

Python 2 vs Python 3

There are two Pythons!

- Currently: Python 2.7 and Python 3.6
- Minimal differences for beginners (except `print`)
- ‘Python 2.x is legacy, Python 3.x is the present and future of the language’

Let’s install Python!

Download Anaconda from

`https://www.anaconda.com/download/`

Using Python

- (I)Python interpreter
- Scripts
- Jupyter (previously IPython) Notebooks

REPL

- **R**ead some input
- **E**valuate
- **P**rint the result
- **L**oop (i.e. repeat)

Let’s try it now: what is **2** **+** **2**?

Python as a calculator

```
1 + 2     # Addition
3 - 4     # Subtraction
5 * 6     # Multiplication
7 / 8     # Division (integer division in Python 2)
7 // 8    # Integer division
9 ** 10   # Exponentiation
11 % 10   # Modulo
```

Careful with operator precedence!

```
1 + 2 ** 3 * 4
1 + (2**3) * 4
1 + ((2**3) * 4)
((1) + (((2)**(3)) * (4)))  # Don't overdo it! :-)
```

Variables and operators

Variables

Variables store values of different types:

- `bool`eans are either `True` or `False`
- `int`egers are whole numbers
- `float`ing points are decimal numbers
- `str`ings are sequences of characters
- `None`

```
session = 'Introduction to Python'  # Or use "..."
day = 26
temperature = 21.2
pressure = 7.6e2  # Same as 7.6 * 10**2 = 760
```

Working with variables

```
1 temp_c = 21.2
2 temp_f = temp_c * 1.8 + 32
```

Working with variables

```
1 temp_c = 21.2
2 temp_f = temp_c * 1.8 + 32
```

Changing type (‘casting’)

```
1 int('26')
2 str(21.2)
3 float('21,2')  # Oops...
```

Operators

- Assignment: `=`
- Comparison: `==`, `!=`, `<`, `<=`, `>`, `>=`
- Mathematical: `+`, `-`, …
- Logical: **`and`**, **`or`**, **`not`**

```
1 + 2
1 < 2
'Lon' + 'don'
'Lon' < 'don'
```

Data structures

Lists

- Lists are ordered sequences of items (so are `str`ings!)
- Not necessarily of the same type

```
1  a = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
2  b = ['I', 'love', 'Python!']
3  a + b
```

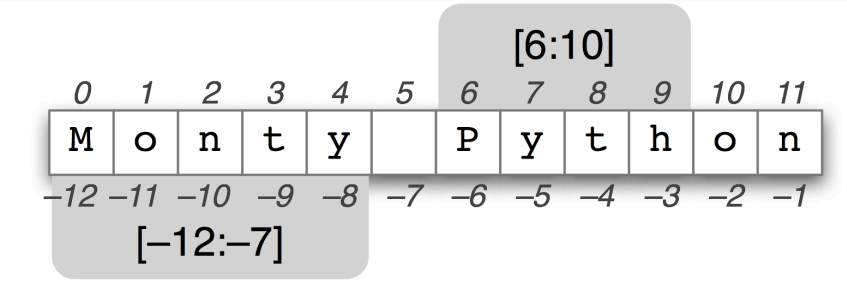
Tuples

- Tuples are similar to lists, but immutable
- Not necessarily of the same type

```
1 a = (0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
2 b = ('I', 'love', 'Python!')
3 a + b
```

Indexing and slicing

```
1 a[0]
2 a[1]
3 a[-2]
4 a[3:]
5 a[:4]
6 a[5:7]
7 a[::2]
```



From *Natural Language Processing with Python*

Dictionaries

- Dictionaries store key–value pairs
- (Not necessarily of the same type)

```
john = {
    'first_name': 'John',
    'last_name': 'Doe',
    'age': 32
}
john['age']
list(john.keys())
list(john.values())
```

Flow control

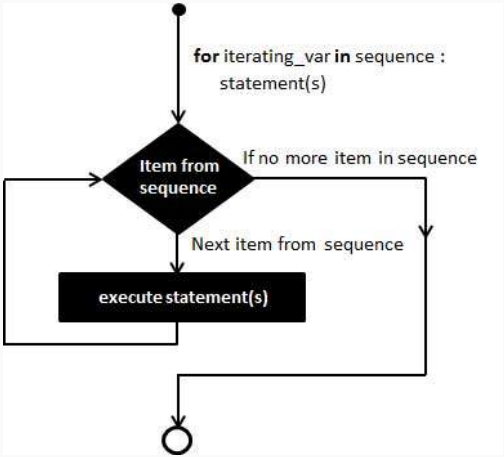
‘If’ statements

- One **`if`** followed by a condition
- Zero or more **`elif`**s (short for ’else if’)
- Optionally a final **`else`**

```
if x < 0:
    print('x is negative')
elif x == 0:
    print('x is zero')
else:
    print('x is positive')
```

‘For’ loops

```
1  for x in range(10):
2      if x == 3:
3          continue
4      elif x == 5:
5          break
6      print(x)
```



From *TutorialsPoint.com*

‘For’ loops with lists

```
1  a = ['one', 'two', 'three', 'four', 'five']
2
3  for value in a:
4      print(value)
5
6  for index, value in enumerate(a):
7      print('Item {} is "{}"'.format(index, value))
```

List comprehensions

```
1  a = [0, 1, 2, 3, 4, 5]
2  b = []
3
4  for x in a:
5      if x > 0:
6          b.append(x**2)
```

```
1  a = [0, 1, 2, 3, 4, 5]
2  b = [x**2 for x in a if x > 0]
```

‘For’ loops with dictionaries

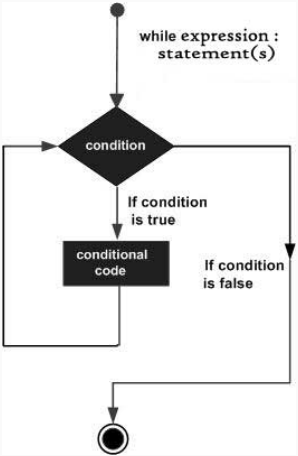
```
a = {
    'one': 'uno',
    'two': 'due',
    'three': 'tre',
    'four': 'quattro',
    'five': 'cinque'
}

for key, value in a.items():
    print('"{}" maps to "{}"'.format(key, value))
```

‘While’ loops

```
1  x = 0
2  while x < 10:
3      x += 1
4      print(x)
```

```
1  x = 0
2  while True:
3      if x == 10:
4          break
5      x += 1
6      print(x)
```



From TutorialsPoint.com

Functions and libraries

Functions

Functions...

- Take input arguments
- Execute statement(s)
- **`return`** output
- (Not necessarily of the same type)

```
1  def multiply(a, b):
2      return a * b
3
4  multiply(2, 3)
```

```
1  def greet(name):
2      print('Hello ' + name)
3
4  greet('John')
```

Libraries

Libraries are…

- Reusable collections of code that someone else (or you) has already written
- Part of the Standard Library (e.g. `random`)
- Distributed through PyPI – the Python Package Index (e.g. `pandas`)

```
1  import random
2
3  random.randint(1, 100)
```

```
1  from random import randint
2
3  randint(1, 100)
```

Making coding more enjoyable

Making coding more enjoyable

Write short lines

- Don’t pack too much in a single long line (historically: 80 characters)
- Clever one-liners don’t make you smarter
- Karma: the next person to read your code will hate you

Making coding more enjoyable

Write short functions

- If it doesn’t fit on screen, break it down
- Think in a modular way (giving birth to programs is painful)
- Write reusable code and don’t repeat yourself

Making coding more enjoyable

Choose meaningful names

- For both functions and variables
- Make the purpose clear
- The code should be the documentation… but write the documentation too

Making coding more enjoyable

Rewrite and polish often

- Understand and accept that you will make mistakes
- There is no royal road to coding
- ‘Negative lines of code’