

Python Programming 101

Gianluca Campanella

Today's class

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

Goals

- Teach you to think like a Computer Scientist
- Make you self-sufficient as a Python beginner

Why Python?

Ideal for scripting and rapid prototyping

- General-purpose, high-level language
- Elegant syntax
- Interpreted
- 'Comes with batteries' (lots of them!)

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'

Let's install Python!

Download Anaconda from

<https://www.anaconda.com/download/>

Using Python

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

Python as a calculator

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

Careful with operator precedence!

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


How to think like a Computer Scientist

Thinking like a Computer Scientist

Computer Scientists...

- Use formal languages to denote ideas
- Design things, assembling components into systems
- Observe the behaviour of complex systems, form hypotheses, and test predictions

What's the most important skill
for a Computer Scientist?

Algorithms

- Step-by-step lists of instructions to solve a problem
- Can be represented in a specific notation (programs)
- Can be executed automatically by a computer

Programs

- Sequences of instructions that describes a computation
- Basic instructions include:
 - Input/output
 - Mathematical and logical operations
 - Conditional execution (if-then)
 - Repetition

Let's write an algorithm!

Compute the sum
of all even numbers
in a given list

Our algorithm... in Python

```
1 def sum_even(numbers):  
2     total = 0  
3     for number in numbers:  
4         if number % 2 == 0:  
5             total += number  
6     return total
```

```
1 sum_even([0, 1, 1, 2, 3, 5, 8, 13, 21, 34])  
2 sum_even(range(101))
```

Variables and operators

Variables

Variables store values of different **types**:

- **booleans** are either **True** or **False**
- **integers** are whole numbers
- **floating points** are decimal numbers
- **strings** are sequences of characters
- **None**

```
1 session = 'Introduction to Python'  # Or use "..."  
2 day = 26  
3 temperature = 21.2  
4 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 1 + 2
2 1 < 2
3 'Lon' + 'don'
4 'Lon' < 'don'
```

Functions and libraries

Functions

Functions...

- Take input **arguments**
- Execute statement(s)
- **return** output

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

Libraries

Libraries are...

- Reusable collections of code
- Part of the Standard Library
- Distributed through PyPI – the Python Package Index

```
1 import random
```

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

```
1 from random import randint
```

```
2  
3 randint(1, 100)
```

Data structures

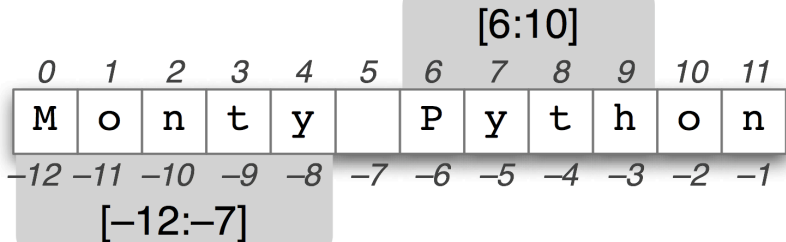
Lists

- Lists are **ordered sequences** of items (so are **strings**!)
- 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

```
1 john = {  
2     'first_name': 'John',  
3     'last_name': 'Doe',  
4     'age': 32  
5 }  
6 john['age']  
7 list(john.keys())  
8 list(john.values())
```

Flow control

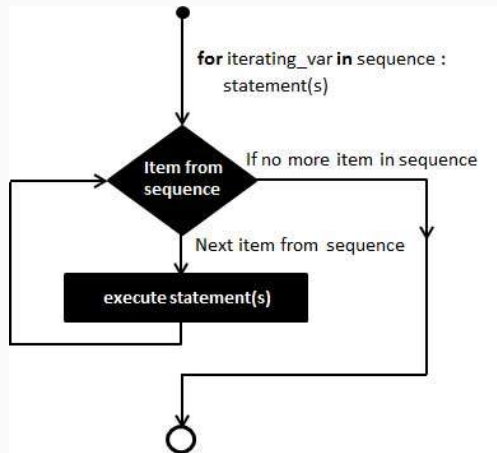
if statements

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

```
1  if x < 0:  
2      print('x is negative')  
3  elif x == 0:  
4      print('x is zero')  
5  else:  
6      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*

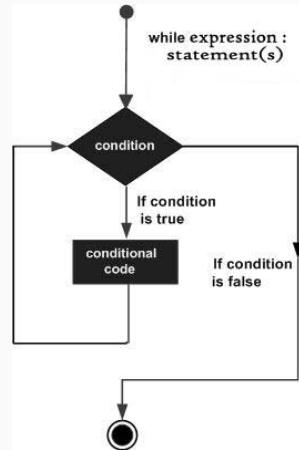
for loops with lists

```
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))
```

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
```



From TutorialsPoint.com

**Making coding
more enjoyable**

Making coding more enjoyable

Write short lines

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

```
1 def primes(n):  
2     return set(range(2, n+1)) - \  
3         set(p*f for p in range(2, int(n**0.5) + 2)  
4             for f in range(2, n//p + 1))
```

Making coding more enjoyable

Write short functions

- If it doesn't fit on screen, break it down
- Think in a modular way
- 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
- Don't be afraid of 'negative lines of code'