

# Lecture 01: Introduction to Python

February 13, 2023



## 1 Introduction to Python

### 1.1 Lists & Dictionaries & Tuples

- Slides by [Ahmed Moustafa](#)
- Content modified from [Pierian Data](#)

Lists can be thought of the most general version of a sequence in Python. Unlike strings, they are **mutable**, i.e. elements inside a list can be changed!

## 2 Creating a list

Lists are constructed with brackets `[ ]` and commas `,` separating every element in the list.

```
[9]: weights = [65.0, 70.5, 72.3, 68.0, 77.2]
      weights
```

```
[9]: [65.0, 70.5, 72.3, 68.0, 77.2]
```

```
[10]: cities = ["London", "Paris", "New York", "Tokyo", "Berlin"]
      cities
```

```
[10]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[19]: len(cities)
```

```
[19]: 5
```

```
[11]: my_list = [1, 2.5, "hello", "world", 42, "python"]
      my_list
```

```
[11]: [1, 2.5, 'hello', 'world', 42, 'python']
```

### 3 Indexing and Slicing

Indexing and slicing work just like in strings:

```
[12]: 'London'
```

```
[13]: cities[1:]
```

```
[13]: ['Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[14]: cities + ["Cairo", "Alexandria"]
```

```
[14]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

### 4 Indexing and Slicing

```
[15]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[16]: cities += ["Cairo", "Alexandria"]
      cities
```

```
[16]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[17]: cities * 2
```

```
[17]: ['London',
      'Paris',
      'New York',
      'Tokyo',
      'Berlin',
      'Cairo',
      'Alexandria',
      'London',
      'Paris',
      'New York',
      'Tokyo',
      'Berlin',
      'Cairo',
      'Alexandria']
```

```
[18]: len(cities)
```

```
[18]: 7
```

## 5 First Things First

As with any programming course, here is the Hello World! in Python.

```
[5]: print ("Hello World!")
```

Hello World!

<IPython.core.display.Image object>

## 6 Variable

A variable is a named storage location used to hold a value. The value of a variable can be changed and it can be used in expressions and operations

## 7 Variable Assignment

- names can not start with a number
- names can not contain spaces, use `_` instead
- names can not contain any of these symbols: `'",<>/?|\!@#%^&*~--+`
- according to Style Guide for Python Code ([PEP8](#)), it's considered best practice that names are lowercase with underscores
- avoid using Python built-in keywords like `list` and `str`
- avoid using the single characters `l` (lowercase letter el), `O` (uppercase letter oh) and `I` (uppercase letter eye) as they can be confused with `1` and `0`

## 8 Dynamic Typing

Python uses *dynamic typing*, meaning you can reassign variables to different data types. This makes Python very flexible in assigning data types; it differs from other languages that are statically typed.

```
[7]: my_cat = 2
     my_cat
```

[7]: 2

```
[8]: my_cat = ['Basbousa', 'Lucy']
     my_cat
```

[8]: ['Basbousa', 'Lucy']

## 9 Pros and Cons of Dynamic Typing

- Pros of Dynamic Typing
  - very easy to work with
  - faster development time
- Cons of Dynamic Typing
  - may result in unexpected bugs!

## 10 Assigning Variables

Variable assignment follows `name = object`, where a single equals sign `=` is an assignment operator

```
[9]: a = 5
     a
```

```
[9]: 5
```

## 11 Reassigning Variables

Python lets you reassign variables with a reference to the same object.

```
[10]: a = a + 10
      a
```

```
[10]: 15
```

There's actually a shortcut for this. Python lets you add, subtract, multiply and divide numbers with reassignment using `+=`, `-=`, `*=`, and `/=`.

```
[11]: a += 10
      a
```

```
[11]: 25
```

```
[12]: a *= 2
      a
```

```
[12]: 50
```

## 12 Determining variable type with `type()`

You can check what type of object is assigned to a variable using Python's built-in `type()` function. Common data types include:

```
[13]: type(a)
```

```
[13]: int
```

## 13 Numbers

Basically there are two types of numbers: - 2 is interger `int` - 2.0 is floating point `float`

Example	Number Type
1,2,-5,1000	Integers
1.2,-0.5,2e2,3E2	Floating point

```
[14]: type(2)
```

```
[14]: int
```

```
[15]: type(2.0)
```

```
[15]: float
```

## 14 Basic Arithmetic 1/2

```
[16]: 2+1 # Addition
```

```
[16]: 3
```

```
[17]: 2-1 # Subtraction
```

```
[17]: 1
```

```
[18]: 2*2 # Multiplication
```

```
[18]: 4
```

```
[19]: 3/2 # Division
```

```
[19]: 1.5
```

## 15 Basic Arithmetic 2/2

```
[20]: 3//2 # Floor division (It returns the result of division rounded down to the  
      ↪nearest integer)
```

```
[20]: 1
```

```
[21]: 2**3 # Powers
```

```
[21]: 8
```

Question: how to calculate the square root of 16?

## 16 Order of Operations

```
[22]: 2 + 10 * 10 + 3
```

```
[22]: 105
```

```
[23]: (2+10) * (10+3)
```

[23]: 156

## 17 Strings

Strings in Python are **text**, such as names, stored as a sequence or a list of characters. For example, Python understands the string 'AUC' to be a sequence of letters in a specific order. This means we will be able to use indexing to grab particular letters (like the first letter A, or the last letter C).

## 18 Creating a String

To create a string in Python you need to use either single quotes ' or double quotes ".

```
[24]: 'Hello'
```

```
[24]: 'Hello'
```

```
[25]: 'Hello World!'
```

```
[25]: 'Hello World!'
```

```
[26]: "This is also a string"
```

```
[26]: 'This is also a string'
```

```
[27]: 'I'm using single quotes, but this will create an error'
```

```
Cell In[27], line 1
    'I'm using single quotes, but this will create an error'
    ~
SyntaxError: invalid syntax
```

```
[28]: 'Now I\'m ready to use the single quotes inside a string!' # Using escape_
      ↪character
```

```
[28]: "Now I'm ready to use the single quotes inside a string!"
```

```
[29]: "Now I'm ready to use the single quotes inside a string!" # Using double quotes
```

```
[29]: "Now I'm ready to use the single quotes inside a string!"
```

## 19 Printing a String

Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a **print** function.

```
[30]: 'Hello World'
```

```
[30]: 'Hello World'
```

```
[31]: 'Hello World 1'  
      'Hello World 2'
```

```
[31]: 'Hello World 2'
```

```
[32]: print('Hello World 1')  
      print('Hello World 2')
```

```
Hello World 1
```

```
Hello World 2
```

```
[33]: print('Hello World 1\nHello World 2') # using \n for new line
```

```
Hello World 1
```

```
Hello World 2
```

## 20 String Indexing 1/3

Since strings are a sequence, we can use brackets [] after an object to call its index. We should also note that indexing **starts at 0** for Python.

```
[34]: name = 'Emma'  
      name
```

```
[34]: 'Emma'
```

```
[35]: name[0]
```

```
[35]: 'E'
```

```
[36]: name[1]
```

```
[36]: 'm'
```

```
[37]: name[-1]
```

```
[37]: 'a'
```

## 21 String Indexing 2/3

```
[38]: name[:2]
```

```
[38]: 'Em'
```

```
[39]: name[2:]
```

```
[39]: 'ma'
```

```
[40]: name[::-1]
```

```
[40]: 'Emma'
```

## 22 String Indexing 3/3

```
[41]: name[:2]
```

```
[41]: 'Em'
```

What will be the output of `name[::-1]`

## 23 String Properties 1/3

String in Python are **immutable** i.e., once a string is created, the elements within it can not be changed or replaced.

```
[42]: name
```

```
[42]: 'Emma'
```

```
[43]: name[0] = 'e'
```

```
-----  
TypeError                                Traceback (most recent call last)  
Cell In[43], line 1  
----> 1 name[0] = 'e'  
  
TypeError: 'str' object does not support item assignment
```

## 24 String Properties 2/3

So if we need to change the value of a string, we will need to **reassign** it the new value:

```
[44]: name = name + " Stone"  
name
```

```
[44]: 'Emma Stone'
```

```
<IPython.core.display.Image object>
```



## 25 String Properties 3/3

```
[46]: name * 5
```

```
[46]: 'Emma StoneEmma StoneEmma StoneEmma StoneEmma Stone'
```

## 26 Bulit-in String Method

In Python, we can call objects' methods with a period and then the method name in the following form: `object.method(parameters)`. And here are some built-in methods in strings:

```
[47]: name.upper() # Convert to upper case
```

```
[47]: 'EMMA STONE'
```

```
[48]: name.lower() # Convert to lower case
```

```
[48]: 'emma stone'
```

```
[49]: name.split() # Split by a separator (the default are white spaces)
```

```
[49]: ['Emma', 'Stone']
```

```
[50]: name.replace("m", "M")
```

```
[50]: 'EMMa Stone'
```

## 27 More Python String Methods

A comprehensive list of string methods in Python can be found:

- here: [Python String Functions at Digital Ocean](#), and
- here: [Python String Methods at Geeks for Geeks](#)

BTW, both are excellent resources for additional documentation and examples.

## 28 Summary

- Python is awesome
- Python uses **dynamic typing**
- Parentheses ( ) are for calling **functions**
- Square brackets [ ] are are indexing **lists**
- Strings are **immutable** lists
- Lists start indexing at **zero**

```
[51]: print("Thank you!")
```

Thank you!

<IPython.core.display.Image object>