

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](#)

2 Lists

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!

3 Creating a list

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

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

```
[2]: [65.0, 70.5, 72.3, 68.0, 77.2]
```

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

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

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

```
[4]: 5
```

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

```
[5]: [1, 2.5, 'hello', 'world', 42, 'python']
```

4 Indexing and Slicing

Indexing and slicing work just like in strings:

```
[6]: 'London'
```

```
[7]: cities[1:]
```

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

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

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

5 Indexing and Slicing

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

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

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

```
[11]: cities * 2
```

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

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

```
[12]: 7
```

6 First Things First

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

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

```
Hello World!
```

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

7 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

8 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

9 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.

```
[15]: my_cat = 2
      my_cat
```

```
[15]: 2
```

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

```
[16]: ['Basbousa', 'Lucy']
```

10 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!

11 Assigning Variables

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

```
[17]: a = 5  
a
```

```
[17]: 5
```

12 Reassigning Variables

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

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

```
[18]: 15
```

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

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

```
[19]: 25
```

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

```
[20]: 50
```

13 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:

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

```
[21]: int
```

14 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

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

```
[22]: int
```

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

```
[23]: float
```

15 Basic Arithmetic 1/2

```
[24]: 2+1 # Addition
```

```
[24]: 3
```

```
[25]: 2-1 # Subtraction
```

```
[25]: 1
```

```
[26]: 2*2 # Multiplication
```

```
[26]: 4
```

```
[27]: 3/2 # Division
```

```
[27]: 1.5
```

16 Basic Arithmetic 2/2

```
[28]: 3//2 # Floor division (It returns the result of division rounded down to the
      ↪nearest integer)
```

```
[28]: 1
```

```
[29]: 2**3 # Powers
```

```
[29]: 8
```

Question: how to calculate the square root of 16?

17 Order of Operations

```
[30]: 2 + 10 * 10 + 3
```

```
[30]: 105
```

```
[31]: (2+10) * (10+3)
```

```
[31]: 156
```

18 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).

19 Creating a String

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

```
[32]: 'Hello'
```

```
[32]: 'Hello'
```

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

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

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

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

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

```
Cell In[35], line 1
    'I'm using single quotes, but this will create an error'
    ^
SyntaxError: invalid syntax
```

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

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

20 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.

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

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

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

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

21 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.

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

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

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

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

22 String Indexing 2/3

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

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

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

23 String Indexing 3/3

```
[ ]: name[::-2]
```

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

24 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.

```
[ ]: name
```

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

25 String Properties 2/3

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

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

26 String Properties 3/3

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

27 Built-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:

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

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

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

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

28 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.

29 Summary

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


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