Python Programming Language

Prepared by: Mohamed Ayman

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

spring 2020

sw.eng.MohamedAyman@gmail.com

facebook.com/cs.MohamedAyman

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

Lecture 2

Variable Types

Course Roadmap

Part 1: Python Basics and Functions

Lecture 1: Python Overview

Lecture 2: Variable Types

Lecture 3: Basic Operations

Lecture 4: Conditions

Lecture 5: Loops

Lecture 6: Functions

Lecture Agenda

We will discuss in this lecture the following topics

1- Python Variables
2- Python Numbers
3- Python Strings
4- Python Lists
5- Python Tuples
6- Python Dictionaries
7- Python Sets
8- Data Type Conversion


Let's
START UP

Lecture Agenda

Section 1: Python Variables

Section 2: Python Numbers

Section 3: Python Strings

Section 4: Python Lists

Section 5: Python Tuples

Section 6: Python Dictionaries

Section 7: Python Sets

Section 8: Data Type Conversion

Python Variable

- A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. You can think variable as a bag to store books in it and those books can be replaced at any time.
- In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.
- One of the most powerful features of a programming language is the ability to manipulate variables. A variable is a name that refers to a value. An assignment statement creates new variables and gives them values.
- A value is one of the basic things a program works with, like a letter or a number. The values we have seen so far are 1, 2, and "Hello, World!", These values belong to different types: 2 is an integer, and "Hello, World!" is a string, so called because it contains a "string" of letters. You (and the interpreter) can identify strings because they are enclosed in quotation marks.

Assigning Values to Variables

- Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory.
- Based on the data type of a variable, the interpreter allocation memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to the variables, you can store integers, decimals or characters in these variables.
- Python Variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

Example:

```
url = 'www.google.com'
print(url)
```

Output:

```
www.google.com
```

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously. Here an integer object is created with the value 1, and the three variable are assigned to the same memory location. You can also assign multiple objects to multiple variables

Example:

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)

x = y = z = "same"

print (x)
print (y)
print (z)
```

Output:

```
5
3.2
Hello

same
same
same
```

Python Constant

- A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later. Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag.

Example:

```
PI = 3.14
GRAVITY = 9.8

print(PI)
print(GRAVITY)
```

Output:

```
3.14
9.8
```

Naming convention for variables and constants

- Create a name that makes sense. Suppose, vowel makes more sense than v.
- Use camelCase notation to declare a variable. It starts with lowercase letter.
  For example: [myName, myAge, myAddress]
- Use underscores to separate between words in a variable name to declare a variable.
  For example: [my_name, my_age, my_address]
- Use capital letters where possible to declare a constant.
  For example: [PI, G, MASS, TEMP]
- Constants are put into Python modules and meant not be changed.
- Never use special symbols like !, @, #, $, %, etc., and don't start name with a digit.
- Constant and variable names would have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
  For example: [snake_case, MACRO_CASE, camelCase, CapWords]

Python Literals

- The data stored in memory can be of many types. For example, a person age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Every value in Python has a data type. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
- Literal is a raw data given in a variable or constant. In Python, there are various types of literals they are as follows:

  Numeric Literals

  String Literals

  Boolean Literals

  Literal Collections

Mutable vs. Immutable in Python

Mutable vs. Immutable in Python

- Every variable in python holds an instance of an object. There are two types of objects in python i.e. Mutable and Immutable objects. Whenever an object is instantiated, it is assigned a unique variable id. The type of the object is defined at the runtime and it can't be changed afterwards. However, it's state can be changed if it is a mutable object.
- To summaries the difference, mutable objects can change their state or contents and immutable objects can't change their state or content.
- Mutable and immutable objects are handled differently in python. Immutable objects are quicker to access and are expensive to change because it involves the creation of a copy. Whereas mutable objects are easy to change. Use of mutable objects is recommended when there is a need to change the size or content of the object.
- Exception: However, there is an exception in immutability as well. We know that tuple in python is immutable. But the tuple consists of a sequence of names with unchangeable bindings to objects.

Lecture Agenda

- ✔ Section 1: Python Variables
- **Section 2: Python Numbers**
- Section 3: Python Strings
- Section 4: Python Lists
- Section 5: Python Tuples
- Section 6: Python Dictionaries
- Section 7: Python Sets
- Section 8: Data Type Conversion

Python Numbers

- Number data type store numeric values. Number objects are created when you assign a value to them.
- Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.
- Python supports three different numerical types

    int (signed integer)

    float (floating point real value)

    complex (complex numbers)

```
a = 5
print(a)

a = 2.0
print(a)

a = 1+2j
print(a)
```

```
5
2.0
(1+2j)
```

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- **Section 3: Python Strings**
- Section 4: Python Lists
- Section 5: Python Tuples
- Section 6: Python Dictionaries
- Section 7: Python Sets
- Section 8: Data Type Conversion

Python Strings

- String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.
- Python allows either pair of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 to the end. The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.
- In Python, Updating or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

Python Strings

Example:	Output:
`x = "Hello Programming"`	
`print(x)`	Hello Programming
`print(len(x))`	17
`print(x[2])`	l
`print(x[2:9])`	llo Pro
`print(x[:4])`	Hell
`print(x[11:])`	amming
`print(x[-1])`	g
`print(x[-5:])`	mming
`print(x[:-7])`	Hello Prog
`print(x[-8:-2])`	grammi
`print(x[4:-2])`	o Programmi
`print(x[-8:14])`	gramm
`print(x[:])`	Hello Programming

H	e	l	l	o		P	r	o	g	r	a	m	m	i	n	g
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Python Strings

Example:

```
x = "Hello Python"
print(x)
print(len(x))
print(x[2:5] + x[9:12])
print(x[:6] + 'World')
print(x * 2)
print(x[:5] * 3)
print(2 * x[6:])
print((x[6:8] + ' ') * 3)
```

Output:

```
Hello Python
12
llohon
Hello World
Hello PythonHello Python
HelloHelloHello
PythonPython
Py Py Py
```

H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Python Strings

Example:

```
x = 'Hello Python'
print(x)
print(len(x))
print(x[::4])
print(x[::3])
print(x[::2])
print(x[::1])
print(x[::-1])
print(x[::-2])
print(x[::-3])
print(x[::-4])
```

Output:

```
Hello Python
12
Hot
HlPh
HloPto
Hello Python
nohtyP olleH
nhy le
nt l
nyl
```

H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Python Strings

Example:

```
x = 'Hello Python'
print(x)
print(len(x))
print(x[3:8:4])
print(x[3:8:3])
print(x[3:8:2])
print(x[3:8:1])
print(x[7:2:-4])
print(x[7:2:-3])
print(x[7:2:-2])
print(x[7:2:-1])
```

Output:

```
Hello Python
12
ly
lP
l y
lo Py
yl
yo
y l
yP ol
```

H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Quiz

- Which of the following statements is valid, according to python string?

```
x = 'Hello World'
```

- [ ] `x = x + '!'`
- [ ] `x[:5] = 'Hi'`
- [ ] `x = x[:5] + ' All'`
- [ ] `x[6:] = 'All'`
- [ ] `x = 'Hi ' + x[6:]`
- [ ] `x[0] = 'M'`
- [ ] `print(x[2:2])`

Quiz Solution

- Which of the following statements is valid, according to python string?

```
x = 'Hello World'
```

- [x] `x = x + '!'`
- [ ] `x[:5] = 'Hi'`
- [x] `x = x[:5] + ' All'`
- [ ] `x[6:] = 'All'`
- [x] `x = 'Hi ' + x[6:]`
- [ ] `x[0] = 'M'`
- [x] `print(x[2:2])`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- **Section 4: Python Lists**
- Section 5: Python Tuples
- Section 6: Python Dictionaries
- Section 7: Python Sets
- Section 8: Data Type Conversion

Python Lists

- List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible. All the items in a list do not need to be of the same type, Declaring a list is pretty straight forward.
- Items separated by commas are enclosed within brackets [ ]. To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data types.
- The value stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.
- Lists need not be homogeneous always which makes it a most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Python Lists

Example:

```
x = ['c++', 123, 'abcd', 2.3, 'python']
print(x)
print(len(x))
print(x[3])
print(x[2:4])
print(x[:4])
print(x[2:])
print(x[-1])
print(x[-4:])
print(x[:-3])
```

Output:

```
['c++', 123, 'abcd', 2.3, 'python']
5
2.3
['abcd', 2.3]
['c++', 123, 'abcd', 2.3]
['abcd', 2.3, 'python']
'python'
[123, 'abcd', 2.3, 'python']
['c++', 123]
```

```
'c++', 123, 'abcd', 2.3, 'python'
0      1     2      3     4
-5     -4    -3     -2    -1
```

Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[1] + x[3])
print(x[0] + x[-1])
print(x[:2] + x[3:])
print(x[2] * 3)
print(x[1] * 3)
print(x[2:4] * 2)
print((x[1:3] + x[4:5]) * 2)
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
2003.2
pythonjava
['python', 2000, 3.2, 'java']
c++c++c++
6000
['c++', 3.2, 'c++', 3.2]
[2000, 'c++', 'java', 2000, 'c++', 'java']
```

```
'python', 2000, 'c++', 3.2, 'java'
0         1     2      3     4
-5        -4    -3     -2    -1
```

Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[::4])
print(x[::3])
print(x[::2])
print(x[::1])
print(x[::-1])
print(x[::-2])
print(x[::-3])
print(x[::-4])
```

```
'python', 2000, 'c++', 3.2, 'java'
0          1     2      3     4
-5         -4    -3     -2    -1
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
['python', 'java']
['python', 3.2]
['python', 'c++', 'java']
['python', 2000, 'c++', 3.2, 'java']
['java', 3.2, 'c++', 2000, 'python']
['java', 'c++', 'python']
['java', 2000]
['java', 'python']
```

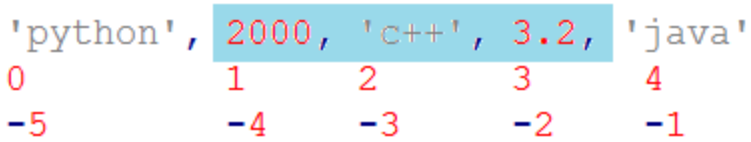
Python Lists

Example:

```
x = ['python', 2000, 'c++', 3.2, 'java']
print(x)
print(len(x))
print(x[1:4:4])
print(x[1:4:3])
print(x[1:4:2])
print(x[1:4:1])
print(x[-2:0:-4])
print(x[-2:0:-3])
print(x[-2:0:-2])
print(x[-2:0:-1])
```

Output:

```
['python', 2000, 'c++', 3.2, 'java']
5
[2000]
[2000]
[2000, 3.2]
[2000, 'c++', 3.2]
[3.2]
[3.2]
[3.2, 2000]
[3.2, 'c++', 2000]
```



Quiz

- Which of the following statements is valid, according to python list?

```
x = [10, 20, 30, 40, 50]
```

- [ ] `x = x + [60]`
- [ ] `x[:3] = [70, 80]`
- [ ] `x = x[:5] + [90, 100]`
- [ ] `x[3:] = [110]`
- [ ] `x = [120] + x[2:]`
- [ ] `x[0] = 130`
- [ ] `print(x[2:2])`

Quiz Solution

- Which of the following statements is valid, according to python list?

```
x = [10, 20, 30, 40, 50]
```

- [x] `x = x + [60]`
- [x] `x[:3] = [70, 80]`
- [x] `x = x[:5] + [90, 100]`
- [x] `x[3:] = [110]`
- [x] `x = [120] + x[2:]`
- [x] `x[0] = 130`
- [x] `print(x[2:2])`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- ✔ Section 4: Python Lists
- **Section 5: Python Tuples**
- Section 6: Python Dictionaries
- Section 7: Python Sets
- Section 8: Data Type Conversion

Python Tuples

- Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.
- It is defined within parentheses ( ) where items are separated by commas. The main difference between lists and tuples is Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are enclosed in parentheses ( ) and cannot be updated. Tuples can be thought of as read-only lists.
- Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list. Tuples are immutable and hence they do not allow deletion of a part of it.

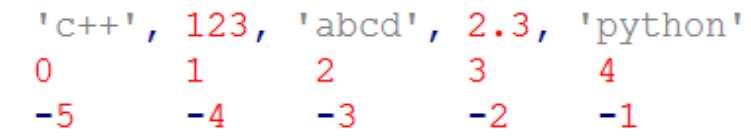
Python Tuples

Example:

```
x = ('c++', 123, 'abcd', 2.3, 'python')
print(x)
print(len(x))
print(x[3])
print(x[2:4])
print(x[:4])
print(x[2:])
print(x[-1])
print(x[-4:])
print(x[:-3])
```

Output:

```
('c++', 123, 'abcd', 2.3, 'python')
5
2.3
('abcd', 2.3)
('c++', 123, 'abcd', 2.3)
('abcd', 2.3, 'python')
'python'
(123, 'abcd', 2.3, 'python')
('c++', 123)
```



Python Tuples

Example:

```
x = ('python', 2000, 'c++', 3.2, 'java')
print(x)
print(len(x))
print(x[1] + x[3])
print(x[0] + x[-1])
print(x[:2] + x[3:])
print(x[2] * 3)
print(x[1] * 3)
print(x[2:4] * 2)
print((x[1:3] + x[4:5]) * 2)
```

Output:

```
('python', 2000, 'c++', 3.2, 'java')
5
2003.2
pythonjava
('python', 2000, 3.2, 'java')
c++c++c++
6000
('c++', 3.2, 'c++', 3.2)
(2000, 'c++', 'java', 2000, 'c++', 'java')
```

'python'	2000	'c++'	3.2	'java'
0	1	2	3	4
-5	-4	-3	-2	-1

Python Tuples

Example:

```
x = ('python', 2000, 'c++', 3.2, 'java')
print(x)
print(len(x))
print(x[::4])
print(x[::3])
print(x[::2])
print(x[::1])
print(x[::-1])
print(x[::-2])
print(x[::-3])
print(x[::-4])
```

Output:

```
('python', 2000, 'c++', 3.2, 'java')
5
('python', 'java')
('python', 3.2)
('python', 'c++', 'java')
('python', 2000, 'c++', 3.2, 'java')
('java', 3.2, 'c++', 2000, 'python')
('java', 'c++', 'python')
('java', 2000)
('java', 'python')
```

```
'python', 2000, 'c++', 3.2, 'java'
0          1     2      3     4
-5         -4    -3     -2    -1
```

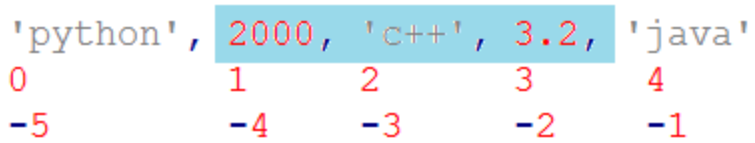
Python Tuples

Example:

```
x = ('python', 2000, 'c++', 3.2, 'java')
print(x)
print(len(x))
print(x[1:4:4])
print(x[1:4:3])
print(x[1:4:2])
print(x[1:4:1])
print(x[-2:0:-4])
print(x[-2:0:-3])
print(x[-2:0:-2])
print(x[-2:0:-1])
```

Output:

```
('python', 2000, 'c++', 3.2, 'java')
5
(2000)
(2000)
(2000, 3.2)
(2000, 'c++', 3.2)
(3.2)
(3.2)
(3.2, 2000)
(3.2, 'c++', 2000)
```



Why Use Tuples?

- Tuples typically store heterogeneous data, similar to how lists typically hold homogeneous data. It's not a hard-coded rule but simply a convention that some Python programmers follow. Because tuples are immutable, they can be used to store different data about a certain thing.
- Tuples are processed faster than lists. If you are creating a constant set of values that won't change, and you need to simply iterate through them, use a tuple.
- The sequences within a tuple are essentially protected from modification. This way, you won't accidentally change the values, nor can someone misuse an API to modify the data. (An API is an application programming interface. It allows programmers to use a program without having to know the details of the whole program). Tuples are used in string formatting, by holding multiple values to be inserted into a string.
- For example, a contact list could conceivably stored within a tuple; you could have a name and address (both strings) plus a phone number (integer) within on data object. The biggest thing to remember is that standard operations like slicing and iteration return new tuple objects. In my programming, I like use lists for everything except when I don't want a collection to change. It cuts down on the number of collections to think about, plus tuples don't let you add new items to them or delete data. You have to make a new tuple in those cases.

Quiz

- Which of the following statements is valid, according to python tuple?

```
x = (10, 20, 30, 40, 50)
```

- [ ] `x = x + (60,)`
- [ ] `x[:3] = (70, 80)`
- [ ] `x = x[:5] + (90, 100)`
- [ ] `x[3:] = (110, 140)`
- [ ] `x = (120,) + x[2:]`
- [ ] `x[0] = 130`
- [ ] `print(x[2:2])`

Quiz Solution

- Which of the following statements is valid, according to python tuple?

```
x = (10, 20, 30, 40, 50)
```

- [x] `x = x + (60,)`
- [ ] `x[:3] = (70, 80)`
- [x] `x = x[:5] + (90, 100)`
- [ ] `x[3:] = (110, 140)`
- [x] `x = (120,) + x[2:]`
- [ ] `x[0] = 130`
- [x] `print(x[2:2])`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- ✔ Section 4: Python Lists
- ✔ Section 5: Python Tuples
- **Section 6: Python Dictionaries**
- Section 7: Python Sets
- Section 8: Data Type Conversion

Python Dictionaries

- Dictionary is an unordered collection of key-value pairs, It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. Python dictionaries are kind of hash-table type. They work like associative arrays or hashes found in perl and consist of key-value pairs.
- In Python, a Dictionary can be created by placing sequence of elements within curly {} braces, separated by 'comma'. Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable.
- Dictionary can also be created by the built-in function dict( ). An empty dictionary can be created by just placing to curly braces {}.

Python Dictionaries

Example:

```
x = {'python':'easy', 'c++':'medium', 'java':'hard'}
print(x)
print(len(x))
print(x['python'])
x['c#'] = 'medium'
x['c++'] = 0
print(x)
print(len(x))
print(x['c#'])
print(x['c++'])
```

Output:

```
{'python':'easy', 'c++':'medium', 'java':'hard'}
3
easy
{'python':'easy', 'c++':0, 'java':'hard', 'c#':'medium'}
4
medium
0
```

Python Dictionaries

Example:

```
x = {'London': 6, 'Paris': 8, 'Barcelone': 11}
print(x)
x['Paris'] = {'Zurich': 2, 'Roma': 5}
x['Barcelone'] = 'windy'
print(x)
print(x['Paris']['Zurich'])
print(x['Barcelone'][:3])
```

Output:

```
{'London': 6, 'Paris': 8, 'Barcelone': 11}
{'London': 6, 'Paris': {'Zurich': 2, 'Roma': 5}, 'Barcelone': 'windy'}
2
win
```

Quiz

- Which of the following statements is valid, according to python dictionary?

```
x = {'London':6, 'Paris':8, 'Barcelone':11}
```

- [ ] `x = x + {'Berlin':4}`
- [ ] `x['Berlin'] = 4`
- [ ] `x = {'Zurich':2, 'Roma':5}`
- [ ] `x['Paris'] = {'Zurich':2, 'Roma':5}`
- [ ] `x['London'] = 7`
- [ ] `print(x[1])`
- [ ] `print(x['Barcelone'])`

Quiz Solution

- Which of the following statements is valid, according to python dictionary?

```
x = {'London':6, 'Paris':8, 'Barcelone':11}
```

- [ ] `x = x + {'Berlin':4}`
- [x] `x['Berlin'] = 4`
- [x] `x = {'Zurich':2, 'Roma':5}`
- [x] `x['Paris'] = {'Zurich':2, 'Roma':5}`
- [x] `x['London'] = 7`
- [ ] `print(x[1])`
- [x] `print(x['Barcelone'])`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- ✔ Section 4: Python Lists
- ✔ Section 5: Python Tuples
- ✔ Section 6: Python Dictionaries
- **Section 7: Python Sets**
- Section 8: Data Type Conversion

Python Sets

- In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. Set in Python is equivalent to sets in mathematics. The order of elements in a set is undefined though it may consist of various elements.
- Elements of a set can be added and deleted, elements of the set can be iterated, various standard operations (union, intersection, difference) can be performed on sets. Besides that, the major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.
- Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Set can also be created by the built-in function set( ). Items in a set are not ordered. We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.

Python Sets

Example:

```
x = {5, 2, 1, 4, 3}
print(x)
print(len(x))

x = {2, 2, 1, 1, 3}
print(x)
print(len(x))

x = {'ab', 'cd', 'ef'}
print(x)
print(len(x))

x = {'ab', 'ef', 'cd', 'ab', 'ef'}
print(x)
print(len(x))
```

Output:

```
{1, 2, 3, 4, 5}
5

{1, 2, 3}
3

{'ab', 'ef', 'cd'}
3

{'ab', 'cd', 'ef'}
3
```

Python Sets

- Python set operations (union, intersection, difference and symmetric difference)

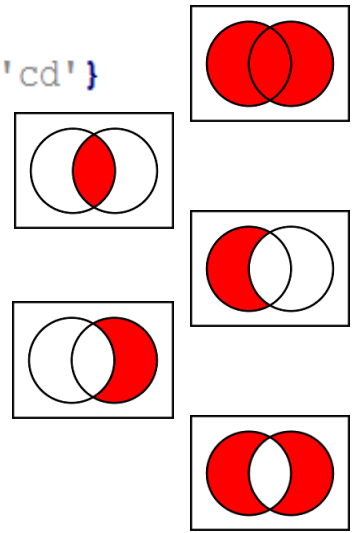
Example:

```
x = {0, 2, 'ab', 4, 6, 'cd', 8}
y = {1, 2, 'ef', 3, 4, 'cd', 5}

print(x | y)
print(x & y)
print(x - y)
print(y - x)
print(x ^ y)
```

Output:

```
{0, 1, 2, 3, 4, 5, 6, 'ef', 8, 'ab', 'cd'}
{2, 'cd', 4}
{0, 8, 'ab', 6}
{1, 3, 5, 'ef'}
{0, 1, 3, 5, 'ef', 6, 8, 'ab'}
```



Quiz

- Which of the following statements is valid, according to python set?

```
x = {0, 2, 4, 6, 8}
```

- [ ] `x = set()`
- [ ] `x = {}`
- [ ] `x = x | {}`
- [ ] `x = {} & x`
- [ ] `print(x[0])`
- [ ] `x = x ^ x`
- [ ] `x = x - set()`

Quiz Solution

- Which of the following statements is valid, according to python set?

```
x = {0, 2, 4, 6, 8}
```

- [x] `x = set()`
- [ ] `x = {}`
- [ ] `x = x | {}`
- [ ] `x = {} & x`
- [ ] `print(x[0])`
- [x] `x = x ^ x`
- [x] `x = x - set()`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- ✔ Section 4: Python Lists
- ✔ Section 5: Python Tuples
- ✔ Section 6: Python Dictionaries
- ✔ Section 7: Python Sets

Section 8: Data Type Conversion

Data Type Conversion

- We can convert between different data types by using different type conversion functions like int(), float(), str() etc. Type Conversion is the conversion of object from one data type to another data type.
- Implicit Type Conversion is automatically performed by the Python interpreter, but Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
- Sometimes, you may need to perform conversions between the built-in types, you simply use the type-name as a function. There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

```
print(float(5))
print(int(10.6))
print(int(-10.6))
print(float('2.5'))
print(str(25))
```

```
5.0
10
-10
2.5
25
```

Data Type Conversion

Function	Description
int(x [,base])	Converts x to an integer. The base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.

Data Type Conversion

Function	Description
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

Data Types Conversion

Example:

```
x, y, z = '11000', '24', 24.0
print(int(x, 2))
print(int(x, 8))
print(int(x, 10))
print(int(x, 16))
print(int(y, 10))
print(float(x))
print(float(y))
print(float(z))
print(str(x))
print(str(y))
print(str(z))
```

Output:

```
24
4608
11000
69632
24
11000.0
24.0
24.0
11000
24
24.0
```

Data Types Conversion

Example:

```
x = [7, 'abc', -4.3, 7]
y = (7, 'abc', -4.3, 7)
z = {7, 'abc', -4.3, 7}
print(list(x))
print(list(y))
print(list(z))
print(tuple(x))
print(tuple(y))
print(tuple(z))
print(set(x))
print(set(y))
print(set(z))
```

Output:

```
[7, 'abc', -4.3, 7]
[7, 'abc', -4.3, 7]
['abc', -4.3, 7]
(7, 'abc', -4.3, 7)
(7, 'abc', -4.3, 7)
('abc', -4.3, 7)
{'abc', -4.3, 7}
{'abc', -4.3, 7}
{'abc', -4.3, 7}
```

Data Types Conversion

Example:

```
x, y, z = 'a', 97, 24
print(ord(x))
print(chr(y))
print(hex(z))
print(oct(z))
print(bin(z))
```

Output:

```
97
a
0x18
0o30
0b11000
```

```
x = 'hello python'
print(list(x))
print(tuple(x))
print(set(x))
```

```
['h', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n']
('h', 'e', 'l', 'l', 'o', ' ', 'p', 'y', 't', 'h', 'o', 'n')
{'h', 'n', 'l', 'y', 't', 'p', 'e', 'o', ' '}
```

Data Types Conversion

Example:

```
print(bool(5))
print(bool(-7))
print(bool(5.1))
print(bool(-7.1))
print(bool(0))
print(bool(None))
print(bool('abc'))
print(bool(''))
print(bool([]))
print(bool(()))
print(bool({}))
print(bool(set()))
print(bool(dict()))
```

Output:

```
True
True
True
True
False
False
True
False
False
False
False
False
False
```

Data Types

Example:

```
x = 5
print(type(x))
x = 7.3
print(type(x))
x = 'python'
print(type(x))
x = [2, 3.2, 'c++']
print(type(x))
x = (2, 3.2, 'c++')
print(type(x))
x = {'math': 92, 'programming': 96}
print(type(x))
x = {3.14, 8, 'java'}
print(type(x))
x = True
print(type(x))
```

Output:

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
<class 'set'>
<class 'bool'>
```

Python Output

- Python Output Using print() function

We use the print() function to output data to the standard output device (screen), we can notice that a space was added between the string and the value of variable a. This is by default, but we can change it.

The actual syntax of the print() function is

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Example:

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='*')
print(1, 2, 3, 4, sep='$', end='&')
```

Output:

```
1 2 3 4
1*2*3*4
1$2$3$4&
```

Python Output

- We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

Example:

```
x = 87.654321
print('The value of x is %.0f' % x)
print('The value of x is %.1f' % x)
print('The value of x is %.2f' % x)
print('The value of x is %.3f' % x)
print('The value of x is %.4f' % x)
```

Output:

```
The value of x is 88
The value of x is 87.7
The value of x is 87.65
The value of x is 87.654
The value of x is 87.6543
```

Python Input

- Up till now, our programs were static. The value of variables were defined or hard coded into the source code. To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

```
input([prompt])
```

where prompt is the string we wish to display on the screen. It is optional.

Python Input

- In the following examples we read a single value with specific data type.

Example:

```
x = input('Enter a string: ')
print(x)
print(type(x))

x = int(input('Enter an int number: '))
print(x)
print(type(x))

x = float(input('Enter a float number: '))
print(x)
print(type(x))
```

Console:

```
Enter a string: Berland
Berland
<class 'str'>

Enter an int number: 10
10
<class 'int'>

Enter a float number: 24.5
24.5
<class 'float'>
```

Python Input

- In the following examples we read multiple values with specific data type.

Example:

```
x, y, z = map(str, input('Enter 3 strings: ').split())
print(x, y, z)
print(type(x), type(y), type(z))

x, y, z = map(int, input('Enter 3 int numbers: ').split())
print(x, y, z)
print(type(x), type(y), type(z))

x, y, z = map(float, input('Enter 3 float numbers: ').split())
print(x, y, z)
print(type(x), type(y), type(z))
```

Console:

```
Enter 3 strings: AB CD EF
AB CD EF
<class 'str'> <class 'str'> <class 'str'>

Enter 3 int numbers: 10 20 30
10 20 30
<class 'int'> <class 'int'> <class 'int'>

Enter 3 float numbers: 3.1 3.2 3.3
3.1 3.2 3.3
<class 'float'> <class 'float'> <class 'float'>
```

Python Input

- In the following example we read multiple values with different data types.

Example:

```
x, y, z = input('Enter 3 values: ').split()
print(x, y, z)
print(type(x), type(y), type(z))
x, y, z = int(x), str(y), float(z)
print(x, y, z)
print(type(x), type(y), type(z))
```

Console:

```
Enter 3 values: 10 Berland 24.5
10 Berland 24.5
<class 'str'> <class 'str'> <class 'str'>

10 Berland 24.5
<class 'int'> <class 'str'> <class 'float'>
```

Quiz

- Which of the following input blocks matched with input statements?

```
1.2 -7.0 5.3
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

```
python_programming_language
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

Quiz Solution

- Which of the following input blocks matched with input statements?

```
1.2 -7.0 5.3
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [x] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

```
python_programming_language
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [x] `x = str(input())`
- [x] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

Quiz

- Which of the following input blocks matched with input statements?

```
python programming language
7.2
```

```
x = int(input())
x = float(input())
x = str(input())
x = input()
x, y, z = map(int, input().split())
x, y, z = map(float, input().split())
x, y, z = map(str, input().split())
x, y, z = input().split()
```

```
x = int(input())
x = float(input())
x = str(input())
x = input()
x, y, z = map(int, input().split())
x, y, z = map(float, input().split())
x, y, z = map(str, input().split())
x, y, z = input().split()
```

Quiz Solution

- Which of the following input blocks matched with input statements?

```
python programming language
7.2
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [x] `x, y, z = map(str, input().split())`
- [x] `x, y, z = input().split()`

- [ ] `x = int(input())`
- [x] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

Quiz

- Which of the following input blocks matched with input statements?

```
-5
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

```
-7 3 11
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

Quiz Solution

- Which of the following input blocks matched with input statements?

```
-5
```

- [x] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [ ] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

```
-7 3 11
```

- [ ] `x = int(input())`
- [ ] `x = float(input())`
- [ ] `x = str(input())`
- [ ] `x = input()`
- [x] `x, y, z = map(int, input().split())`
- [ ] `x, y, z = map(float, input().split())`
- [ ] `x, y, z = map(str, input().split())`
- [ ] `x, y, z = input().split()`

Lecture Agenda

- ✔ Section 1: Python Variables
- ✔ Section 2: Python Numbers
- ✔ Section 3: Python Strings
- ✔ Section 4: Python Lists
- ✔ Section 5: Python Tuples
- ✔ Section 6: Python Dictionaries
- ✔ Section 7: Python Sets
- ✔ Section 8: Data Type Conversion

Practice

Problem 1

- Take as an input the name, age, weight of user in separate lines then print them in the same line.

Such that the name is string, age is integer and weight is float.

Test Cases:

Test Case 1:

```
Mark
24
70.2

Mark 24 70.2
```

Test Case 2:

```
Petr
26
85.6

Petr 26 85.6
```

Test Case 3:

```
Paul
28
96.7

Paul 28 96.7
```

Problem Solution

- Take as an input the name, age, weight of user in separate lines then print them in the same line.

Such that the name is string, age is integer and weight is float.

Test Cases:

Test Case 1:

```
Mark
24
70.2

Mark 24 70.2
```

Test Case 2:

```
Petr
26
85.6

Petr 26 85.6
```

Test Case 3:

```
Paul
28
96.7

Paul 28 96.7
```

```
name = str(input())
age = int(input())
weight = float(input())
print(name, age, weight)
```

Problem 2

- Take as an input the degree of 3 courses in your school in the same line then print them in separate lines.

Such that the degree of courses are integers.

Test Cases:

Test Case 1:

```
92 94 89

92
94
89
```

Test Case 2:

```
85 87 82

85
87
82
```

Test Case 3:

```
78 81 75

78
81
75
```

Problem Solution

- Take as an input the degree of 3 courses in your school in the same line then print them in separate lines.

Such that the degree of courses are integers.

Test Cases:

Test Case 1:	Test Case 2:	Test Case 3:
92 94 89	85 87 82	78 81 75
92 94 89	85 87 82	78 81 75

```
course1, course2, course3 = map(int, input().split())
print(course1)
print(course2)
print(course3)
```

Problem 3

- Take as an input the name, age, weight of user in the same line then print them in separate lines.

Such that the name is string, age is integer and weight is float.

Test Cases:

Test Case 1:

```
Mark 24 70.2

Mark
24
70.2
```

Test Case 2:

```
Petr 26 85.6

Petr
26
85.6
```

Test Case 3:

```
Paul 28 96.7

Paul
28
96.7
```

Problem Solution

- Take as an input the name, age, weight of user in the same line then print them in separate lines.

Such that the name is string, age is integer and weight is float.

Test Cases:

Test Case 1:

```
Mark 24 70.2

Mark
24
70.2
```

Test Case 2:

```
Petr 26 85.6

Petr
26
85.6
```

Test Case 3:

```
Paul 28 96.7

Paul
28
96.7
```

```
name, age, weight = input().split()
age = int(age)
weight = float(weight)
print(name, age, weight, sep='\n')
```


DO
MORE.