
# Programming in Python

**Prof. Sudarshan Iyengar**
Department of Computer Science and Engineering
Indian Institute of Technology Ropar

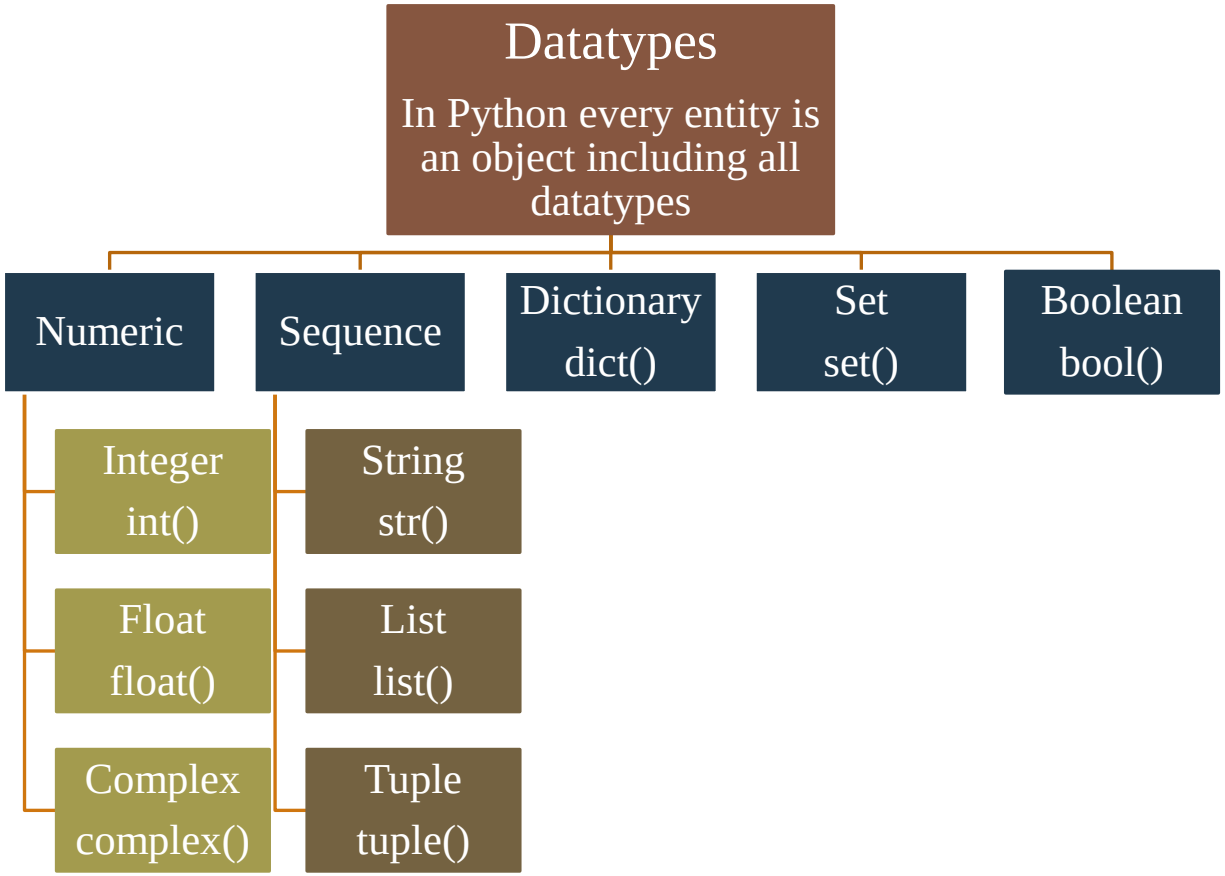
**Mr. Omkar Joshi**
Course Instructor
IITM Online Degree Programme


# Programming in Python

## Course summary

| Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|
| 1. print()<br>2. Variables<br>3. input()<br>4. Data types<br>5. Operators<br>6. Expressions | 1. Strings<br>2. Types of quotes and Escape characters<br>3. if-elif-else<br>4. Types of import statements | 1. while loop<br>2. for loop<br>3. Formatted printing<br>4. Nested loops<br>5. break, continue, pass | 1. Lists<br>2. Nested lists<br>3. Obvious sort<br>4. Matrix operations | 1. Functions<br>2. Types of functions<br>3. Types of function arguments<br>4. Scope of variable | 1. Lists<br>2. Tuples<br>3. Dictionaries<br>4. Sets |

| Week 8 | Week 9 | Week 10 | Week 11 |
|---|---|---|---|
| 1. Recursion<br>2. Binary search | 1. File handling<br>2. Pandas library | 1. Object Oriented Programming<br>2. Numpy library<br>3. Matplotlib library | 1. Exception handling<br>2. Functional programming |




## Variables

No need to declare variables in Python. It is created when the value is assigned to it first time. Python allows dynamic typing. Variable name is any sequence of characters a-z, A-Z, 0-9 and underscore but not starting with numbers. Variables are case sensitive.

## Operators

| Operators | Symbol / Keyword | Name / Description |
|---|---|---|
| **Arithmetic** | + | Addition |
| | - | Subtraction |
| | * | Multiplication |
| | / | Division |
| | % | Modulus |
| | // | Floor division |
| | ** | Exponential |
| **Comparison** | == | Double equal to |
| | != | Not equal to |
| | > | Greater than |
| | < | Less than |
| | >= | Greater than equal to |
| | <= | Less than equal to |
| **Logical** | and | Returns True if both statements are true |
| | or | Returns False if one of the statements are True |
| | not | Reverse the result |
| **Membership** | in | Returns True if the value is present in the sequence |
| | not in | Returns True if the value is not present in the sequence |
| **Identity** | is | Returns True if both variables point to the same object |
| | is not | Returns True if both variable do not point to the same object |

## Formatted printing

It allows programmer to take more control over printing output rather than simply printing space separated values.

1. **f – string:**

```
name = input()
print(f'Hi, {name}!')
```

2. **format():**

```
name = input()
print('Hi, {}!'.format(name))
```

## Escape characters

Backslash '\' is considered as escape character in Python. It is used to insert characters in strings which are illegal otherwise.

| Escape character | Result |
|---|---|
| \\ | Backslash |
| \' | Single quote |
| \" | Double quotes |
| \n | New line |
| \t | Tab |

## Types of import statements

1. `import math`
   It will import the math library

2. `from math import *`
   It will import the entire contents of math library

3. `from math import pi`
   It will import only variable pi from math library

4. `import calendar as cal`
   It will import the calendar library and it can be accessed as cal

5. `from calendar import month as m`
   It will import only month method from calendar library and it can be accessed as m

## Useful external libraries

Pandas, NumPy, Matplotlib

# Conditional statements

These conditional statements are used when some kind of decision making is required. Comparison operators are most useful when we use these conditional statements.

## if block:

The goal of this statement is to check if the given condition is True or False. If it is True then Python will execute the following indented lines of code.

```
if a > b:
    print('a is greater than b')
```

## if – elif blocks:

elif stands for else if. This is Python's way of saying, if the first condition is False then it will check the next condition.

```
if a > b:
    print('a is greater than b')
elif a < b:
    print('a is less than b')
```

## if – elif – else blocks:

If all the conditions given in if and/or elif evaluates to be False then Python executes else block.

```
if a > b:
    print('a is greater than b')
elif a < b:
    print('a is less than b')
else:
    print('a and b are equal')
```

## Inline if – elif – else (Ternary operator):

The above code can be written in a single line as well.

```
print('a is greater than b' if a > b else
'a is less than b' if a < b else 'a and b
are equal')
```

## Nested conditional statements:

Any of the above mentioned conditional blocks can be written inside any other conditional blocks and such kind of structure is referred as nested if – else.

# Loops

## while loop:

It enables you to execute set of statements as long as the given condition is True.

```
i = 0
while i < 10:
    print(i)
    i += 1
```

This code will print numbers from 0 to 9.

## for loop:

1. **Using range():**

```
for i in range(10):
    print(i)
```

This code is equivalent to the above while loop and it will also print numbers from 0 to 9.

2. **Without using range():**

```
for i in 'Hello all. Welcome to Python':
    print(i)
```

This code will print the given string one character at a time in every iteration.

## Nested loops:

Any of the above mentioned loops (while and 2 versions of for) can be written inside any of these loops and such kind of structure is referred as nested iterations/loops.

## Loop control statements:

Loops are used to automate repetitive statements. But sometimes there may arise a condition where we may want to terminate the loop, skip an iteration or ignore that condition. In such conditions we use loop control statements like **break**, **continue** and **pass** respectively.

## range(start, end, step):

It is an in-built function which returns a sequence of numbers based on the values of start, end and step.

**start:** An integer number specifying at which position to start.
It is an optional argument. Default value is 0.

**end:** An integer number specifying at which position to stop (non-inclusive).

**step:** An integer number specifying the incrementation/decrementation.
It is an optional argument. Default value is 1.

## In-built functions

**print():** Prints argument(s) on to the console
**input():** Takes input from the console as a string
**type():** Returns the type of an object
**len():** Returns the length of an object
**max():** Returns the largest value in the iterable
**min():** Returns the smallest value in the iterable
**sum():** Sums the elements in an iterator
**sorted():** Returns the sorted list
**iter():** Returns an iterator object
**next():** Returns the next element in the iterator object
**enumerate():** Returns an enumerate object by adding counter to an iterable
**zip():** Returns an iterator after coupling two or more iterators
**map():** Returns the iterator after applying specific function on each element in it
**filter():** Returns an iterator after applying specific filtering function

## User defined functions

Along with in-built functions, Python allows you to define your own functions called user defined functions. Function is a block of code which executes only when it is called. Function can have parameter(s) which is/are used to store passed argument(s) from the call. It can return a value back to its call as a result. Function is defined using def keyword.

### How to define a function?

```
def add(x, y):
    return x + y
```

### How to call a functions

```
result = add(10, 20)
```

## Types of function arguments

1. **Positional arguments:** Mapping of arguments and parameters happens as per the sequence

```
def myFunction(x, y, z):
    return x + y - z
result = myFunction(10, 20, 30)
```

2. **Keyword arguments:** Mapping of arguments and parameters is given explicitly

```
def myFunction(z, y, x):
    return x + y - z
result = myFunction(x = 10, z = 30, y = 20)
```

3. **Default arguments:** Mapping of arguments and parameters is either explicit or positional but if argument is missing then the default value is considered. Default arguments should be assigned at the end of the list of parameters.

```
def myFunction(z, y = 20, x = 10):
    return x + y - z
result = myFunction(30)
```

| Property | List<br>Video link | Tuple<br>Video link | Dictionary<br>Video link | Set<br>Video link |
|---|---|---|---|---|
| **Notation** | [ ] | ( ) | {'Key': 'Value'} | { } |
| **Creation** | list() | tuple() | dict() | set() |
| **Mutability** | Mutable | Immutable | Mutable | Mutable |
| **Type of elements which can be stored** | Any | Any | Keys: Hashable<br>Values: Any | Hashable |
| **Order of elements** | Ordered | Ordered | Unordered* | Unordered |
| **Duplicate elements** | Allowed | Allowed | Keys: Not allowed<br>Values: Allowed | Not allowed |
| **Operations** | Add, Update, Delete | None | Keys: Add, Delete<br>Values: Add, Update, Delete | Add, Delete |
| **Operations** | Indexing, Slicing, Iteration | Indexing, Slicing, Iteration | Iteration | Iteration |
| **Sorting** | Possible | Not possible | Possible | Not possible |

*Python 3.6 and earlier. Dictionaries are ordered as per Python 3.7 and above.

## Recursive function

When a function calls itself in its definition then it is called as recursive function

```
def add_N_numbers(n):
    if n == 1:
        return n
    else:
        return n + add_N_numbers(n - 1)
result = add_N_numbers(5)
```

## lambda function

It is an anonymous function which can have any number of arguments but can execute only one expression.

```
f = lambda a, b, c: a + b - c
result = f(10, 20, 5)
```

## List comprehension

It is an optimized way to create a list in a single line.

```
myList = [x ** 2 for x in range(10)
if x % 2 == 0]
```

## Exception handling

Errors which occur during execution of the Python program are termed as exceptions. Exceptions are different from syntax errors. Python provides try – except blocks in order to handle such exceptions.

```
try:
    print(a / b)
    print(myDictionary['xyz'])
except ZeroDivisionError:
    print('Denominator cannot be zero')
except KeyError:
    print('This key is not there in the
dictionary')
```

The above code is syntactically correct and will execute successfully if value of variable b is non-zero and ‘xyz’ is a key in myDictionary. But at the same time it will throw an exception if either variable b is zero or myDictionary does not have the key ‘xyz’.

Complete list of in-built exceptions.

### finally block:

It is a special type of code block which always executes after successful execution as well as after termination due to execution. Therefore it is used to deallocate the system resources used in the program, e.g. file pointers.

# File handling

We store our entire information on computer systems using files. Hence, there has to be some way to open/read/write files using Python.

**How to open/create a file?**

It is done using open() function which takes to parameters, first is file name with its extension and second is mode for opening file.

```
f = open('abc.txt', 'r')
```

| Operation | Parameter | Description |
|---|---|---|
| **Create** | ‘x’ | Creates the specified file, returns an error if the file exists |
| **Read** | ‘r’ | Default value. Opens a file for reading, error if the file does not exist |
| **Append** | ‘a’ | Opens a file for appending, creates the file if it does not exist |
| **Write** | ‘w’ | Opens a file for writing, creates the file if it does not exist |
| **Text** | ‘t’ | Default value. Text mode |
| **Binary** | ‘b’ | Binary mode (e.g. images) |

**How to read a file?**

There are three major functions through which we can read files.

**f.read():** It reads the specified number of bytes from the file. The default/optional argument is -1 which indicated the entire file content.

**f.readline():** It reads the file one line at a time.

**f.readlines():** It reads the entire file as a list of strings representing individual lines.

**How to write a file?**

In order to write on to the file, we must open it using either ‘w’ or ‘a’ mode. And, then we can use **f.write()** function to write content.

# Object Oriented Programming

Python is an object oriented programming language. Hence, every entity in Python is an object with its associated attributes and methods.

**How to create class?**

```
class myClass:
def __init__(self):
   pass
```

**How to create object?**

```
obj = myClass()
```

**How to add attributes and methods to class?**

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print(self.name, self.age)

s1 = Student('ABC', 10)
s1.display()
```

**How to implement inheritance?**

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print(self.name, self.age)

class Student(Person):
    def __init__(self, name, age, marks):
        super().__init__(name, age)
        self.marks = marks
    def display(self):
        super().display()
        print(self.marks)

s1 = Student('ABC', 10, 90)
s1.display()
```

**What are the types of inheritance?**

Simple, Hierarchical, Multiple, Multilevel and Hybrid.

# Transition and Use of Python