Shri Ramdeobaba College of Engineering and Management, Nagpur

Department of Electronics Engineering

Digital Image Processing (ENT 355-3)

Name: Prajwal Pandurang Shette

Roll No: B1- 12

Experiment No: 01

Aim: Introduction of python: operators, data types, list, dictionaries, Python If…Else, While, For loops, python functions

Introduction of Python: Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.
- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

Operators:

Operator	Name	Description	Syntax	Example
+	Addition	Performs addition	c = a + b	a = 5, b = 5 then c = 10
-	Subtraction	Performs subtraction	c = a – b	a = 5, b = 3 then c = 2
*	Multiplication	Performs multiplication	c = a * b	a = 5, b = 5 then c = 25
/	Division	Performs division	c = a / b	a = 10, b = 5 then c = 2
%	Modulus	Performs division but returns the remainder	c = a % b	a = 15, b = 2 then c = 1
//	Floor Division	Performs division but returns the quotient in which the digits after the decimal points are removed	c = a // b	a = 15, b = 2 then c = 7
**	Exponent	Performs multiplication to power raised	c = a ** b	a = 2, b = 4 then c = 16

Data types: In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type: `str`

Numeric Types: `int, float, complex`

Sequence Types: `list, tuple, range`

Mapping Type: `dict`

Set Types: `set, frozenset`

Boolean Type: `bool`

Binary Types: `bytes, bytearray, memoryview`

None Type: `NoneType`

Python List: Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

List items are ordered, changeable, and allow duplicate values.List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Ex:
```
thislist=["apple", "banana", "cherry"]
    print(thislist)
```

Ex:
```
thislist=["apple", "banana", "cherry"]
    print(len(thislist))
```

Python Dictionaries: Dictionaries are used to store data values in key: value pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Dictionaries are written with curly brackets, and have keys and values.
Dictionary items are ordered, changeable, and does not allow duplicates.
Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

Ex:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Ex:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

Python If…Else: Python supports the usual logical conditions from mathematics:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions can be used in several ways, most commonly in "if statements" and loops.

An "if statement" is written by using the `if` keyword.

Ex:

```
a = 33
b = 200
if b > a:
  print(“b is greater than a”)
```

Ex:

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

while loop: With the `while` loop we can execute a set of statements as long as a condition is true.

The `while` loop requires relevant variables to be ready, in this example we need to define an indexing variable, `i`, which we set to 1.

Ex:
```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Ex:
```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

For loops: A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Ex:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Ex:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

Ex:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
```

```
    continue
  print(x)
```

Python functions:

A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function.A function can return data as a result.

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Ex:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Ex:

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\nRecursion Example Results")
tri_recursion(6)
```