Introduction to Python

Dr. Indrajeet Kumar

What is Python?

- Python is a popular programming language.

- It is used for:

  web development (server-side),

  software development,

  mathematics,

  system scripting.

What can Python do?

- 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**.

Why Python?

- Works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Introduction

- Two variations:

  IDLE (GUI),
  python (command line)

- Type statements or expressions at prompt:

```
>>> print "Hello, world"
Hello, world
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```

Python Variables

- Variables are containers for storing data values.
- **Creating Variables**

  Python has no command for declaring a variable.

  A variable is created the moment you first assign a value to it.

```
X=5
y="John“
print(x)
print(y)
```

Casting

- If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

You can get the data type of a variable with the type() function.

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Variables

- One Value to Multiple Variables

  x = y = z = "Orange“

- Unpack a Collection

  fruits = ["apple", "banana", "cherry"]
  x, y, z = fruits

- Output Variables

  x = "awesome"
  print("Python is " + x)

Strings

- "hello"+"world" "helloworld" # concatenation
- "hello"*3 "hellohellohello“ # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "hello" < "jello" 1 # comparison
- "e" in "hello“ 1 # search

Lists

- Lists are used to store multiple items in a single variable.
- Lists are created using square brackets:

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

List items are indexed and you can access them by referring to the index number:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

Negative Indexing

Negative indexing means start from the end.
-1 refers to the last item, -2 refers to the second last item etc.

Range of Indexes

```
thislist ="apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Python Tuples

- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and **unchangeable**.
- Tuples are written with round brackets.

  Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Python Dictionaries

- Dictionaries are used to store data values in key: value pairs.

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

- A dictionary is a collection which is ordered*, changeable and does not allow duplicates.

Python Conditions

- Python supports the usual logical conditions.

  Example:

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

Control Structures

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

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

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Ternary Operators

One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

Logical operator

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions
is True")
```

Nested If

```
x = 41
if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Python Loops

- Python has two primitive loop commands:
  - **while loops**
  - **for loops**

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Break statement

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Continue statement

```
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**).

  for *var* in *sequence*:

  statements

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

Looping Through a String

```
for x in "banana":
  print(x)
```

```
for x in range(2, 30, 3):
  print(x)
```

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

Creating a Function

```
def my_function():
  print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

==========

```
def my_function():
  print("Hello from a function")
my_function()
```

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2
= "Tobias", child3 = "Linus")
```

Passing a List as an Argument

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

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

my_function(fruits)
```

Recursion

- Python also accepts function recursion, which means a defined function can call itself.
- It means that a function calls itself.
- This has the benefit of meaning that you can loop through data to reach a result.

Example

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

print("\n\nRecursion Example Results")
test_recursion(6)
```

Thank you