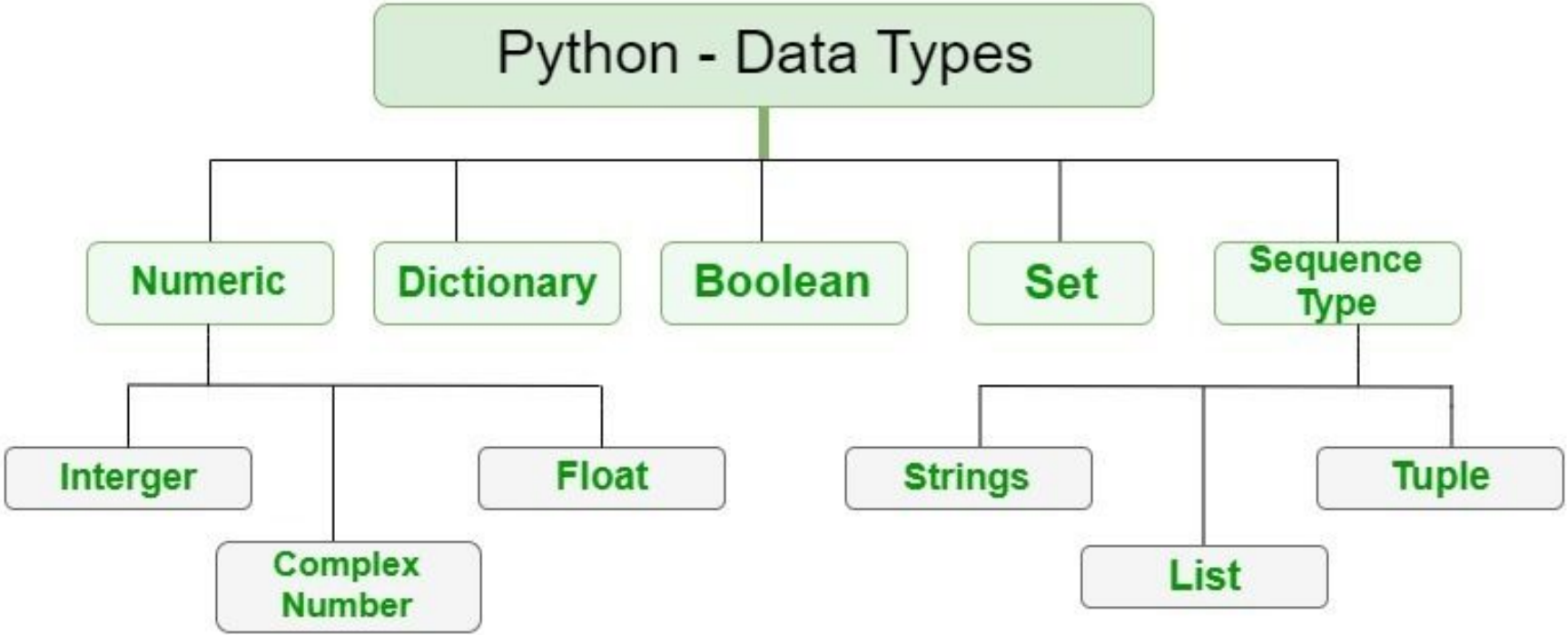
Python Programming

Session - 3

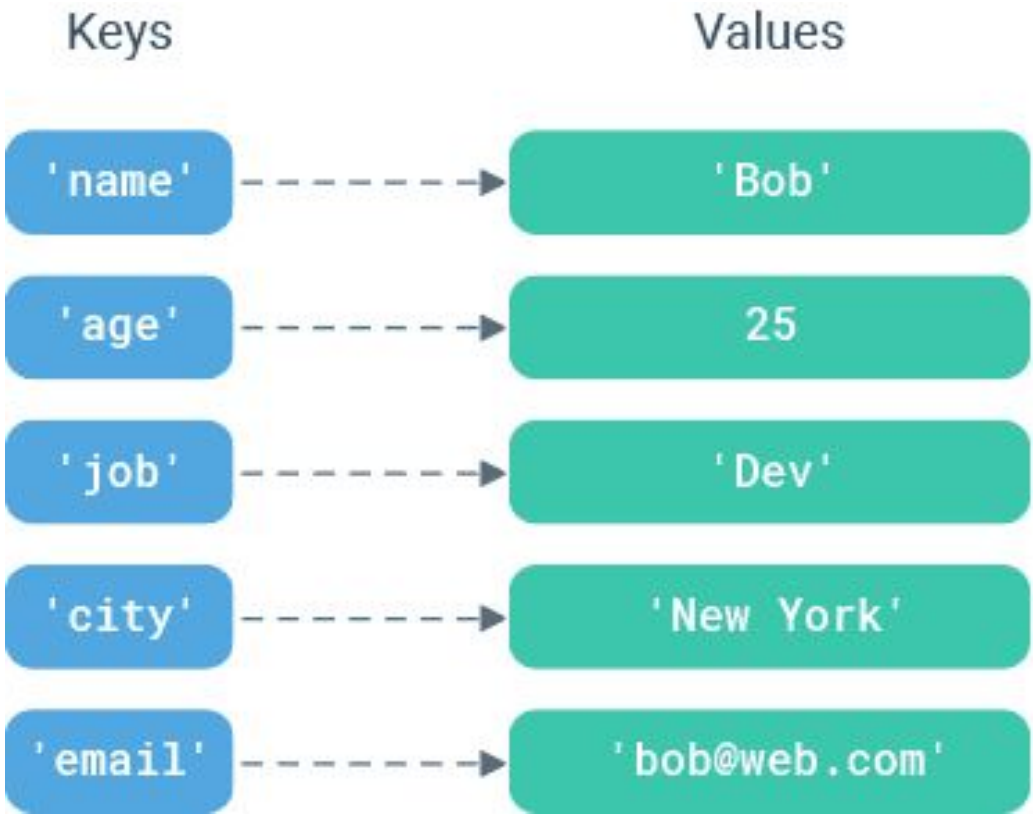
- Ajay Kumar



Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

Table of Difference between List, Set and Tuple

LIST	SET	TUPLE
Lists is Mutable - [ ]	Set is Mutable - { }	Tuple is Immutable - ( )
It is Ordered collection of items	It is Unordered collection of items	It is ordered collection of items
Items in list can be replaced or changed	Duplicate items are not allowed	Items in tuple cannot be changed or replaced

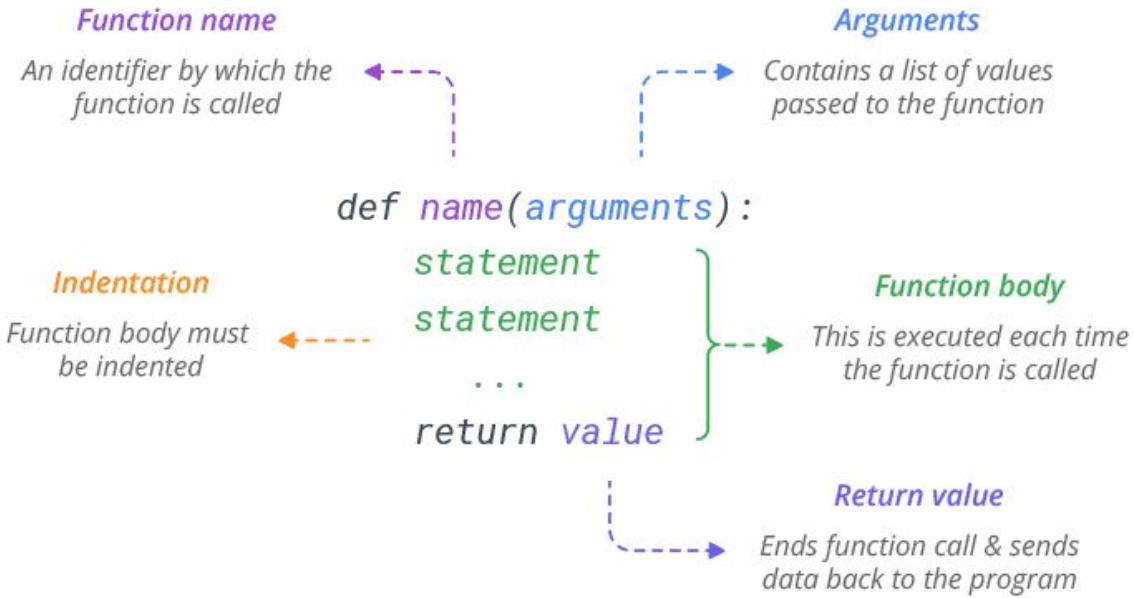


Session Break

GitHub Link

Functions

- Functions are the first step to code reuse.
- They allow you to define a reusable block of code that can be used repeatedly in a program.
- Python provides several built-in functions.
- You can also define your own functions to use within your programs.



Functions

Call a Function

The **def** statement only creates a function but does not call it. After the def has run, you can can call (run) the function by adding parentheses after the function's name.

```
def hello():

    print('Hello, World!')

hello()

   # Prints Hello, World!
```

Pass Arguments

You can send information to a function by passing values, known as arguments. Arguments are declared after the function name in parentheses.

```
# Pass single argument to a function

def hello(name):

    print('Hello,', name)

hello('Bob')

# Prints Hello, Bob
```

Functions

Types of Arguments

It supports multiple types of arguments in the function definition.

- Positional Arguments - `func('Bob', 'developer')`
- Keyword Arguments - `func(name='Bob', job='developer')`
- Default Arguments - `def func(name, job='developer'):`
- Variable Length Positional Arguments (*args) - `print_arguments(1, 54, 60, 8, 98, 12)`
- Variable Length Keyword Arguments (**kwargs) - `print_arguments(name='Bob', age=25)`

Functions

Return Value

To return a value from a function, simply use a return statement. Once a return statement is executed, nothing else in the function body is executed.

```
# Return sum of two values
def sum(a, b):
    return a + b

x = sum(3, 4)
print(x)
    # Prints 7
```

Docstring

You can attach documentation to a function definition by including a string literal just after the function header. Docstrings are usually triple quoted to allow for multi-line descriptions.

```
def hello():
    """This function prints
        message on the screen"""
    print('Hello, World!')
```

Functions

Nested Functions

A Nested function is a function defined within another function. They are useful when performing complex task multiple times within another function, to avoid loops or code duplication.

```
def outer(a, b):
    def inner(c, d):
        return c + d
    return inner(a, b)
result = outer(2, 4)
print(result)
   # Prints 6
```

Recursion

A recursive function is a function that calls itself and repeats its behavior until some condition is met to return a result.

```
def countdown(num):
    if num <= 0:
        print('Stop')
    else:
        print(num)
        countdown(num-1)
countdown(5)
```

Functions

Assigning Functions to Variables

When Python runs a def statement, it creates a new function object and assigns it to the function's name. You can assign a different name to it anytime and call through the new name.

```
def hello():
    print('Hello, World!')
hi = hello
hi()
```

```
def findSquare(x):
    return x ** 2
def findCube(x):
    return x ** 3
# Create a dictionary of functions
exponent = {'square': findSquare, 'cube': findCube}
print(exponent['square'](3))
# Prints 9
print(exponent['cube'](3))
   # Prints 27
```

Functions

Python Function Executes at Runtime

Because Python treats def as an executable statement, it can appear anywhere a normal statement can.

For example you can nest a function inside an if statement to select between alternative definitions.

```
x = 0
if x:
    def hello():
        print('Hello, World!')
else:
    def hello():
        print('Hello, Universe!')

hello()
# Prints Hello, Universe!
```

Session Break

GitHub Link

Variable Scope

Variable Scope

```
# enclosing function
def f1():
    x = 42
    # nested function
    def f2():
        x = 0
        print(x)    # x is 0
    f2()
    print(x)        # x is still 42

f1()
```

```
# enclosing function
def f1():
    x = 42
    # nested function
    def f2():
        nonlocal x
        x = 0
        print(x)    # x is now 0
    f2()
    print(x)        # x remains 0

f1()
```

Lambda Functions

```
lambda parameters: expression
```

```
doubler = lambda x: x*2

print(doubler(2))
# Prints 4

print(doubler(5))
# Prints 10
```

No Statements Allowed

Single Expression Only

IIFE - `print((lambda x: x*2)(3))`
`# Prints 6`

Multiple Arguments

Ways to Pass Arguments

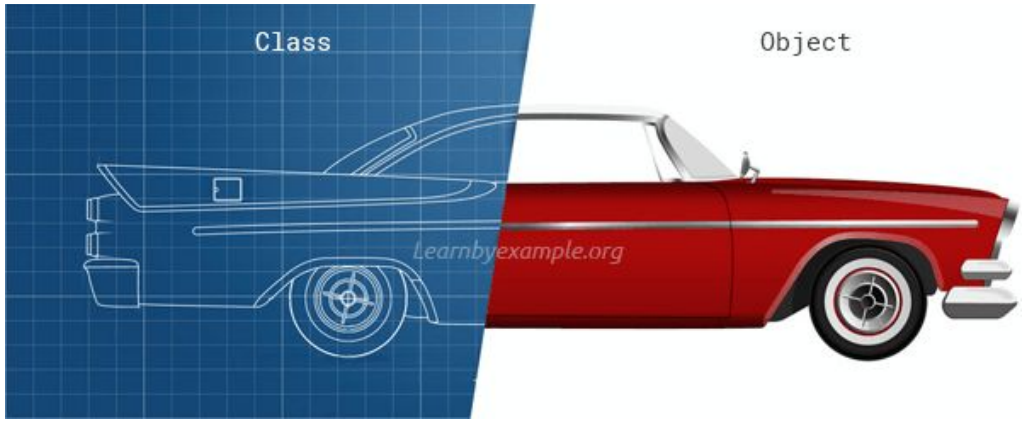
if else in a Lambda

Return Multiple Values

Classes and Objects

Classes and objects are the two main aspects of object-oriented programming.

A class is the blueprint from which individual objects are created. In the real world, for example, there may be thousands of cars in existence, all of the same make and model



Classes and Objects

- Create a Class
- The __init__() Method
- The self Parameter
- Attributes
  - Instance Attribute
  - Class Attribute
- Create an Object
- Access and Modify Attributes
- Methods
  - Instance Methods
  - Class Methods
- Delete Attributes and Objects

Session Break

GitHub Link