Python Fundamentals

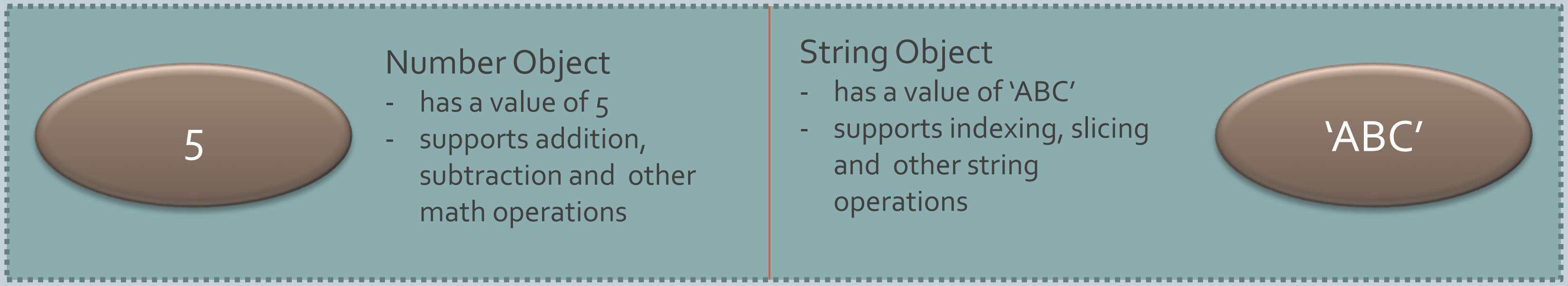
2. LAYING THE FOUNDATION

Laying the Foundation - Objectives

- Python Objects
- Python Application Hierarchy
- Statement v/s Expression
- Python’s Core Data Types
- Variable/Object Creation
- Dynamic Typing in Python
- Mutable v/s Immutable
- Shared References

Python Objects

- Programming is about processing Information for desired results
  - Information = Data
  - Processing = Operation
- In Python data takes the form of objects,
  - Built-in objects which comes with Python or
  - User defined objects
- Everything in Python is just an Object
- Each object is allocated a memory
- Every object has a certain value
- Every object has a set of supported operations

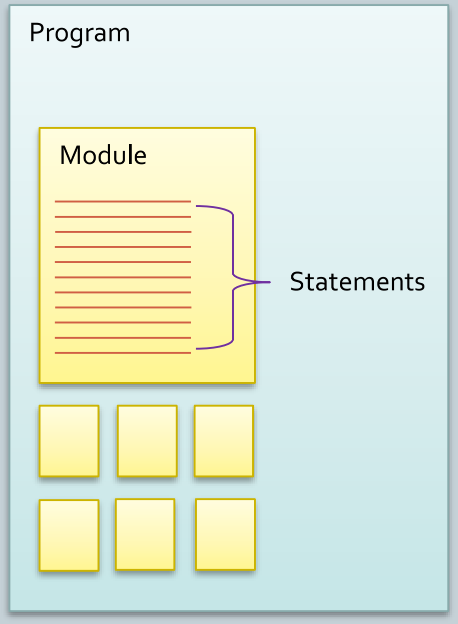


Number Object
- has a value of 5
- supports addition, subtraction and other math operations

String Object
- has a value of 'ABC'
- supports indexing, slicing and other string operations

Python Application Hierarchy

- Python application follows the below hierarchy
- Python programs are made up of one more mode modules
- Modules contains Python statements
- Statements are made up of expressions
  - Statement is a complete line of code that performs some action
- Expressions create and process objects
  - Expressions is the part of code which evaluates to a value



Statement v/s Expression

- A statement is a complete line of code that perform some action whereas an expression is any section of the code that evaluates to a value
- As a thumb rule, if you can print it or assign it to a variable its an expression else it's a statement

e.g. of expression

```
2+2
round(81.5)
'$' + 'Dollar'
None
True
False
```

e.g. of Statement

```
if 5==5:
elif name = 'Guido':
else:
while True:
for each in range(10):
try:
def test():
```

- In Python 3, print is a function which is an expression and you can assign the result of a print function to any variable in Python 3.
- In Python 2, print is an statement which means it does not evaluate to anything

Python’s Core Data Types

- Following data types come built in with Python and are referred as core data types

Data Types	Syntax and example
Numbers	999, 1000, 2.145
Strings	‘Python’ , “Adam’s”, u’\x23python’
Booleans	True, False
List	[1,2,[3,’Name’]], [], list(range(10)
Tuples	(1, 2, ‘Name’) , tuple(‘python’)
Dictionaries	{}, {name: ‘Mark’, role: ’developer’}, dict(name=‘Mark’)
Files	Open(‘foo.txt’, ‘r’)
Sets	{‘x’, ‘y’, ‘z’} , set(‘abc’)

- *Remember everything in Python is object even Functions, modules, etc

Variable & Object creation

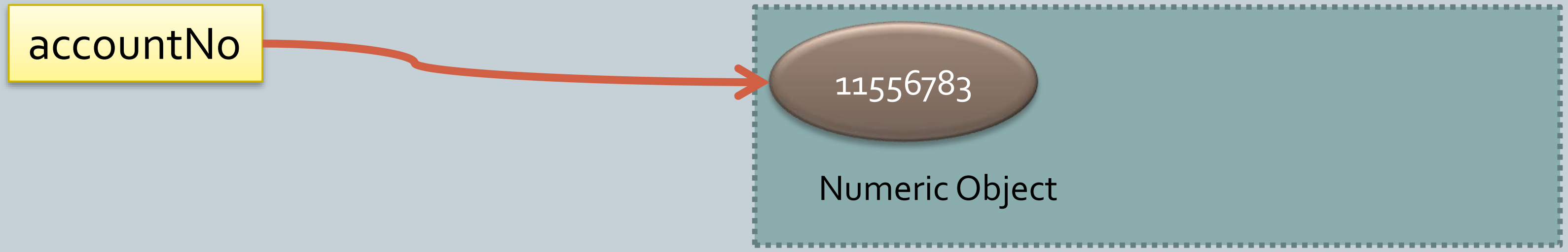
- Objects in Python are created in two ways:
  - By using the literal e.g. 5 , 3.14 , ‘Google’
  - By using the expression that generates a specific object e.g. [1,2], list(‘abc’), {‘a’:1, ‘b’:2} , dict(a=1, b=2), str(‘Python’)
- Reference of an object can be stored in a variable using assignment operator i.e. ‘=‘

```
language = 'Python'
accountNo = 11556783
ingredientList = ['Flour', 'Yeast']
```

- To refer back to the object, same reference variable can be used

```
print(accountNo)
11556783
```

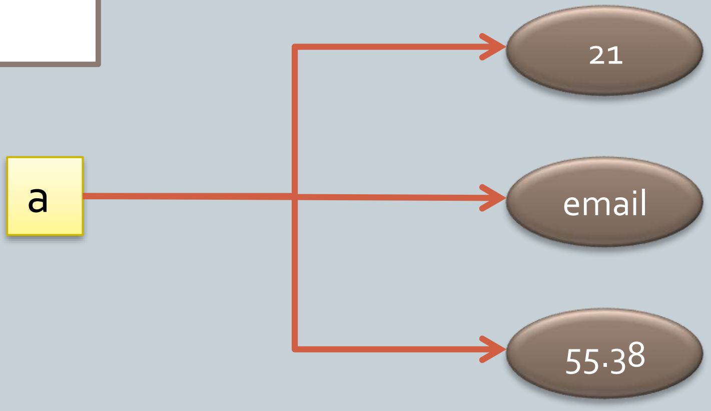
- Reference variable does not store any value. Its just a reference to the object



Dynamic Typing in Python

- Python does not require to specify the type of the variable
- Python determines the type at run time instead of declaration
- Python does not require the declaration of variable ahead of their use
- When you write an expression like `accountNo = 11556783`, Python ;
  - Create an object which represents the value of 11556783
  - Create the variable accountNo if it does not exist
  - Link the variable accountNo to refer to new object 11556783
- Dynamic typing in play

```
a = 21
a = 'email'
a = 55.38
```



Mutable v/s Immutable

- Immutable object is an object that can’t be changed after its created
- Number , String and Tuples are immutable objects in Python

```
name = 'John'; age=32
name = 'Johnathan'
```

new String object ‘Jonathan’ is created and **name** refers to the object

New numeric object 32 is created and age refers to the object

```
name[2]: = 'a'
```

Above will give error as in place changes are not allowed for immutable objects, however;

```
name = name[:2] + 'shua'
print(name)
'Joshua'
```

You can create a new string object with slice and concatenation and assign to same variable

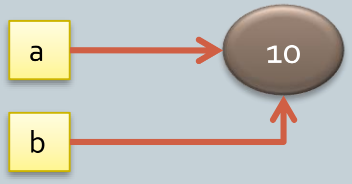
- List, Dictionaries and Sets are mutable i.e. their value can be changed in place

e.g. of In place changing of mutable objects

Shared References

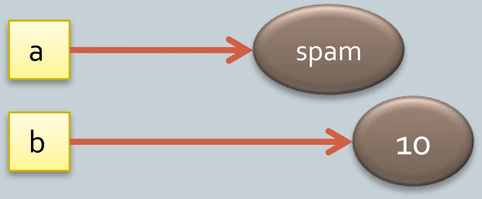
- When multiple variables point to same object, its known as shared reference e.g.

```
a = 10
b = a
```



- In Python, all assignments create and assign the reference to the name
- Changing a to refer to a string object 'spam' will have no effect on b

```
a = 'spam'
```



- If the object being shared is mutable, then in place change to that object using one variable will reflect when you access the object via second variable.
- `'IS'` operator is used to test of shared refrence whereas equals to operator `'=='` is used to test object equality