PYTHON..

BY TECHOLAS TECHNOLOGIES

INTRODUCTION TO PYTHON

- Python is a popular programming language.
- It is used for web development(server-side),software development,system scripting ,data science etc.
- It is derived from many other languages including C,C++ etc.

FEATURES

- Easy-to-learn
- Easy-to-read
- Easy-to-maintain
- Portable
- Interpreted language
- Support automatic garbage collector
- Scalable
- Works on different platforms
- Simple syntax

PYTHON INSTALLATION AND PYCHARM

Python interpreter is free, and downloads are available for all major platforms (Windows, Mac OS, and Linux) in the form of *source* and *binary*. You can download it from the Python Website: python.org.

For downloading pycharm:

https://www.jetbrains.com/pycharm/download/

VARIABLES IN PYTHON

Variables are containers for storing data values.

Unlike other programming languages, Python has no command for declaring a variable.

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

eg:a=19

b=”hello”

Print (a)

print(b)

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT....

Variable Names

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

CONT..

- Variables do not need to be declared with any particular type and can even change type after they have been set.
- String variables can be declared either by using single or double quotes.
- Python allows you to assign values to multiple variables in one line.

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

PRINT() IN PYTHON

The print() function prints the specified message to the screen.

Syntax: print(*object(s)*, separator=*separator*, end=*end*, file=*file*, flush=*flush*)

object(s)-Any object, and as many as you like.

separator-Optional. Specify how to separate the objects

end-Optional. Specify what to print at the end.

file-Optional. An object with a write method

flush-Optional.A Boolean, specifying if the output is flushed or buffered

LIST IN PYTHON

There are four collection data types in the Python programming language:

- List
- Tuple
- Set
- Dictionary

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

CNT...

- A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).
- Also, a list can even have another list as an item.
- List can have duplicate values.
- Items can be accessed with index.

CNT...

LIST-Creation

```
#empty list
  my_list = []
# list of integers
  my_list = [1, 2, 3]
# list with mixed data types
  my_list = [1, "Hello", 3.4]
#nested list
  my_list = ["mouse", [8, 4, 6], ['a']]
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

How to access elements from a list?

1.List Index

We can use the index operator [] to access an item in a list.

Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

- Trying to access an element other than that this will raise an IndexError.
- The index must be an integer.We can't use float or other types,this will result into TypeError.
- Nested list are accessed using nested indexing.

CNT...

my_list = ['p','r','o','b','e']
print(my_list[0]) - Output: p
print(my_list[2]) - Output: o
print(my_list[4]) - Output: e
my_list[4.0] – Output : Error! Only integer can be used for indexing
n_list = ["Happy", [2,0,1,5]] – Nested list
print(n_list[0][1]) - Output: a
print(n_list[1][3]) – Output 5

CNT...

2.Negative indexing

- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_list = ['p','r','o','b','e']
print(my_list[-1]) - Output: e
print(my_list[-5]) – Output p
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

How to slice lists in Python?

- We can access a range of items in a list by using the slicing operator(colon).

```
my_list = ['p','r','o','g','r','a','m','i','z']
print(my_list[2:5])  #index 2nd to 4th  (4th=n-1)
Output : ['o', 'g', 'r']
print(my_list[:-5])  #index 0 to 3rd
Output : ['p', 'r', 'o', 'g']
print(my_list[5:])  #index 5th to end
Output : ['a', 'm', 'i', 'z']
print(my_list[:])  #all elements
Output : ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

How to change or add elements to a list?

- List are mutable, meaning, their elements can be changed unlike string or tuple.
- We can use assignment operator (=) to change an item or a range of items.

```
odd = [2, 4, 6, 8]

odd[0] = 1  # change the 1st item

print(odd)
```

Output: [1, 4, 6, 8]

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

```
odd[1:4] = [3, 5, 7]  # change 2nd to 4th items
print(odd)
Output: [1, 3, 5, 7]
```

append() – add one item to a list at the end

extend() – add several items to a list at the end

len() – to find the length of a list

```
odd = [1, 3, 5]
odd.append(7)
print(odd)
Output: [1, 3, 5, 7]
```

```
odd.extend([9, 11, 13])
print(odd)
Output: [1, 3, 5, 7, 9, 11, 13]
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

•We can also use + operator to combine two lists. This is also called concatenation.

•The * operator repeats a list for the given number of times.

```
odd = [1, 3, 5]
print(odd + [9, 7, 5])
```

Output: [1, 3, 5, 9, 7, 5]

```
print(["re"] * 3)
```

Output: ["re", "re", "re“]

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- insert() – insert an item in a desired location
- insert multiple items by squeezing it into an empty slice of a list.

```
odd = [1, 9]

odd.insert(1,3)
print(odd)
Output: [1, 3, 9]
```

```
odd[2:2] = [5, 7]
print(odd)
Output: [1, 3, 5, 7, 9]
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

```
del my_list[1:5]   # delete multiple items (index 1 to 4)
print(my_list)
Output: ['p', 'm']
del my_list   # delete entire list
print(my_list)  # Error: List not defined
```

- remove() – to remove a given item
- pop() - to remove an item at the given index.(pop() removes and returns the last item if index is not provided)

CNT...

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')
print(my_list)
  # Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list.pop(1))
  # Output: 'o‘
print(my_list)
  # Output: ['r', 'b', 'l', 'e', 'm']
print(my_list.pop())
  # Output: 'm‘
```

```
print(my_list)
 # Output: ['r', 'b', 'l', 'e']
my_list.clear()
print(my_list)
  # Output: []
```

CNT...

we can also delete items in a list by assigning an empty list to a slice of elements.

```
my_list = ['p','r','o','b','l','e','m']
my_list[2:3] = []
my_list
  Output : ['p', 'r', 'b', 'l', 'e', 'm']
my_list[2:5] = []
my_list
  Output : ['p', 'r', 'm']
```

CNT...

How to delete or remove elements from a list?

- Delete one or more items using the keyword ‘del’.
- This can also delete the entire list.

```
my_list = ['p','r','o','b','l','e','m']
del my_list[2]  #delete one item in the index position 2
print(my_list)
Output: ['p', 'r', 'b', 'l', 'e', 'm']
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

More List Operations

```
a = range(5)  # [0,1,2,3,4]
a.append(5)  # [0,1,2,3,4,5]
a.pop()  # [0,1,2,3,4]
a.insert(0, 42)  # [42,0,1,2,3,4]
a.pop(0)  # [0,1,2,3,4]
a.reverse()  # [4,3,2,1,0]
a.sort()  # [0,1,2,3,4]
```

CNT...

my_list = [3, 8, 1, 6, 0, 8, 4]

- print(my_list.index(8)) #returns the first matching index position of the specified item

  Output : 1

- print(my_list.count(8)) #returns the no. of occurrence of the item

  Output : 2

- my_list.sort()

  Output : [0, 1, 3, 4, 6, 8, 8]

- my_list.reverse()

  Output : [8, 8, 6, 4, 3, 1, 0]

TUPLES

- A tuple in Python is similar to a list.
- The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas, in a list, elements can be changed.
- A tuple is a collection which is ordered and unchangeable.
- In Python tuples are written with round brackets ().

CNT...

Creating Tuples

•A tuple is created by placing all the items (elements) inside parentheses (), separated by commas.

•The parentheses are optional, however, it is a good practice to use them.

•A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

CNT...

•Empty tuple

Ømy_tuple = ()

Øprint(my_tuple)

Output: ()

•Tuple having integers

Ømy_tuple = (1, 2, 3)

Øprint(my_tuple)

Output: (1, 2, 3)

•tuple with mixed data types

Ømy_tuple = (1, "Hello", 3.4)

Øprint(my_tuple)

Output: (1, "Hello", 3.4)

•nested tuple

Øprint(my_tuple)

Output: ("mouse", [8, 4, 6], (1, 2, 3))

TECHOLAS

TECHNOLOGY DEMYSTIFIED

CNT..

How to access tuple elements?

1.Indexing

Use the index operator [] to access an item in a tuple where the index starts from 0.

A tuple having 6 elements will have indices from 0 to 5.

- Trying to access an element outside of tuple will raise an IndexError.
- The index must be an integer.We can’t use float or other types,this will result into TypeError.
- Nested tuples are accessed using nested indexing.

CNT...

my_tuple = ('p','e','r','m','i','t')

- print(my_tuple[0])

  Output : 'p'
- print(my_tuple[5])

  Output : 't‘
- print(my_tuple[6])

  Output : IndexError: list index out of range

CNT...

- my_tuple[2.0]

  Output : TypeError: list indices must be integers, not float
- n_tuple = ("mouse", [8, 4, 6], (1, 2, 3)) #nested tuple
  print(n_tuple[0][3]) # nested index

  Output : ‘s’
  print(n_tuple[1][1])

  Output : 4

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

2. Negative Indexing

•The index of -1 refers to the last item, -2 to the second last item and so on.

•Same as that of list

3. Slicing

•We can access a range of items in a tuple by using the slicing operator colon ":".

•This is also similar to list.

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Changing a Tuple

- Unlike lists, tuples are immutable.
- This means that elements of a tuple cannot be changed once it has been assigned.
- But, if the element is itself a mutable datatype like list, its nested items can be changed.
- We can also assign a tuple to different values (reassignment).

CNT...

my_tuple = (4, 2, 3, [6, 5])

- my_tuple[1] = 9

  Output :-TypeError: 'tuple' object does not support item assignment
- my_tuple[3][0] = 9 #However, item of mutable element can be changed

  print(my_tuple)

  Output: (4, 2, 3, [9, 5])
- my_tuple = ('p','r','o','g','r','a','m','i','z') #Tuples can be reassigned

  print(my_tuple)

  Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

CNT...

- Concatenation – use + operator to combine two tuples
- We can also repeat the elements in a tuple for a given number of times using the * operator

```
print((1, 2, 3) + (4, 5, 6))  # Concatenation
```

Output : (1, 2, 3, 4, 5, 6)

```
print(("Repeat",) * 3)
```

Output : ('Repeat', 'Repeat', 'Repeat')

SET

A set is a collection which is ordered and unindexed.In python sets are written with curly brackets.

eg:set1={“apple”,”banana”,”cherry”}

print(set1)

Output:{‘cherry,’apple’,’banana’}

- Sets are unordered,so you cannot be sure in which order of the items will appear.

CNT...

How to access elements in a set?

You cannot access items in a set by referring to an index,since sets are unordered the items has no index.

But you can loop through the set items using a for loop or ask if a specified value is present in a set,by using the in keyword

eg:set1={“apple”,”banana”,”cherry”)

```
For x in set:

    print(x)
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- set1={“abc”,”qwe”,”xyz”}

  print(“qwe” in set1)#checking qwe is present in set1

  output:True

- set2={“wee”,”klj”,”per”}

  print(“abc” in set2)

  output:False

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT....

How to change elements in a set?

Once a set is created,you cannot change its items,but you can add new items.

How to add elements in a set?

To add one element to a set use the add() method.

To add more than one element to a set use the update() method.

CNT...

my_set={1,3}

- my_set.add(2)#adding element 2 in my_set

  print(my_set)

  Output:{1,2,3)

- my_set.update([2,3,4])#adding list using update

  print(my_set)

  Output:{1,2,3,4}

CNT...

- my_set.update([4,5],{1,6,8})#adding list and set

  print(my_set)

  output:{1,2,3,4,5,6,8}

- my_set1={“sads”,”wdww”,”bghd”}

  my_set1.update([“cdf”,”wds”])

  print(my_set1)

  output:{‘bghd’,wds’,’cdf’,’wdww’,’sads’}

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

How to get the length of a set?

To determine how many items a set has,use the len() method.

set1={“xyz”,”abc”}

print(len(set1))

Output:2

CNT...

How to remove an element from a set?

To remove an element in a set,use the remove(),or the discard() method.

- The difference between the two is that,while using discard() if the item does not exist in the set,it remains unchanged.
- But remove() will raise an error in such condition.

CNT...

set1={1,3,4,5,6}

- set1.discard(4)

  print(set1)

  Output:{1,3,5,6}

- set1.remove(2)

  print(set1)

  Output:Error 2 not in the set

CNT...

Similarly we can remove and return an item using pop() method.

s={1,2,3)

print(s.pop())

Output:2

Set be unordered,there is no way of determining which item will be popped.It is completely arbitrary

CNT...

We can also remove all items from a set using clear()

s={1,2,3,4,5}

s.clear()

print(s)

Output:set()

CNT...

Python set operations

Sets can be used to carry out mathematical set operations like union,intersection,difference and symmetric difference.We can do this with operators or methods.

Lets consider the following two sets for the following operations

a={1,2,3,4,5}

b={4,5,6,7,8}

CNT...

- Set Union

  Union of a and b is a set of all elements from both sets.

  Union is performed using | operator.Same can be accomplished using the method union().

  print(a|b)

  Output:{1,2,3,4,5,6,7,8}

  print(a.union(b))

  output:{1,2,3,4,5,6,7,8}

CNT...

- Set Intersection

  Intersection of and b is a set of elements that are common in both sets.

  Intersection is performed using & operator.Same can be accomplished using the method intersection().

  print(a&b)

  Output:{4,5}

  print(a.intersection(b))

  output:{4,5}

CNT...

- Set Difference

Difference of a and b (a - b) is a set of elements that are only in a but not in b.Similarly, b-a is a set of element in b but not in a.

Difference is performed using - operator. Same can be accomplished using the method difference().

print(a-b)

Output:{1,2,3}

print(a.difference(b))

Output:{1,2,3}

CNT...

- Set Symmetric Difference

Symmetric Difference of a and b is a set of elements in both a and b except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric_difference().

print(a^b)

Output:{1,2,3,6,7,8}

print(a.symmetric_difference(b))

output:{1,2,3,6,7,8}

CNT...

Set Methods

Python has a set of built-in methods that you can use on sets.

add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set

CNT...

discard()	Remove the specified item
difference_update()	Removes the items in this set that are also included in another, specified set
intersection_update()	Updates the set with the intersection of itself and another
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another

CNT...

isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitary set element. Raise KeyError if the set is empty

CNT...

Built-in Functions with Set

all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.

CNT...

len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.

CNT...

sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Return the sum of all elements in the set.

DICTIONARY

Dictionary is an ordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair.

Dictionaries are optimized to retrieve values when the key is known.

thisdict = {"brand": "Ford","model": "Mustang","year": 1964}

print(thisdict)

output:{‘brand’:’Ford’,’model’:’mustang’,’year’:1964’}

CNT....

How to create a dictionary?

Creating a dictionary is as simple as placing items inside curly braces {}

separated by comma.

An item has a key and the corresponding value expressed as a pair, key: value.

CNT...

- While values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

  my_dict = {'name':'Jack', 'age': 26}

- update value

  print(my_dict['age']) = 27

  Output: {'name':'Jack', 'age': 27}

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- add item

  my_dict['address'] = 'Downtown'

  print(my_dict)

  Output: { 'name': 'Jack', 'age': 27, ’address': 'Downtown'}

CNT...

How to delete or remove elements from a dictionary?

We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.

- The method, popitem() can be used to remove and return the last item (key, value) from the dictionary.
- All the items can be removed at once using the clear() method.

CNT...

- We can also use the del keyword to remove individual items or the entire dictionary itself.

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

- print(squares.pop(4))# remove a particular item

  Output: 16

- print(squares)

  Output: {1: 1, 2: 4, 3: 9, 5: 25}

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT..

- print(squares.popitem())# removes the last item

  Output: (5, 25)

- print(squares)

  Output: {1:1, 2: 4, 3: 9}

- del squares[1] # delete a particular item

  print(squares)

  Output: {2: 4, 3: 9}

CNT...

- squares.clear()# remove all items

  print(squares)

  Output: {}

- del squares# delete the dictionary itself

CNT...

Dictionary Methods

Method	Description
clear()	Remove all items from the dictionary.
copy()	Return a shallow copy of the dictionary.
dict.fromkeys(seq, [v])	Return a new dictionary with keys from seq and value equal to v (defaults to None).

CNT...

get(key,[d])	Return the value of key. If key doesnot exist, return d (defaults to None).
items()	Return a new view of the dictionary's items (key, value).
keys()	Return a new view of the dictionary's keys.

CNT...

pop(key,[d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem()	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key,[d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None).

CNT...

update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Return a new view of the dictionary's values

DATA TYPES

The data stored in memory can be of many types.

For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

CNT...

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

CNT...

Text type

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator

CNT...

str = 'Hello World!'

print str# Prints complete string

Output:Hello World!

print str[0] # Prints first character of the string

Output:H

print str[2:5] # Prints characters starting from 3rd to 5th

Output:llo

CNT...

print str[2:] # Prints string starting from 3rd character

Output:llo World!

print str * 2 # Prints string two times

Output:Hello World!Hello World!

print str + "TEST"# Prints concatenated string

Output:Hello World!TEST

CNT...

You can get the data type of any object by using the type() function.

Numeric type

Integers, floating point numbers and complex numbers falls under Python numeric type category. They are defined as int, float and complex class in Python.

CNT...

Integers can be of any length, it is only limited by the memory available.

A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here is an example.

CNT...

```
c = 2 + 4j
print(type(c))

c1 = complex(1,2)
print(type(c1))
print(c1)
```

CNT...

Sequence type

Sequences are one of the principal built-in data types besides numerics, mappings, files, instances and exceptions.

- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- The range() type returns an immutable sequence of numbers between the given start integer to the stop integer.

CNT...

Mapping type

The mapping objects are used to map hash table values to arbitrary objects. In python there is mapping type called dictionary. It is mutable.

The keys of the dictionary are arbitrary. As the value, we can use different kind of elements like lists, integers or any other mutable type objects.

CNT...

Set type

In Python,

- Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

CNT...

- Frozen set is just an immutable version of a Python set object. While elements of a set can be modified at any time, elements of frozen set remains the same after creation. Due to this, frozen sets can be used as key in Dictionary or as element of another set. But like sets, it is not ordered (the elements can be set at any index).

```
a = [4,8,2]
frozen = frozenset(a)
frozen.add(7) # generates error since a frozenset can't be mutated
```

CNT...

Boolean type

The bool() method is used to return or convert a value to a Boolean value i.e., True or False, using the standard truth testing procedure.

Syntax:

```
bool([x])
```

CNT...

The bool() method in general takes only one parameter(here x), on which the standard truth testing procedure can be applied. If no parameter is passed, then by default it returns False. So, passing a parameter is optional. It can return one of the two values.

- It returns True if the parameter or value passed is True.
- It returns False if the parameter or value passed is False.

CNT...

Binary type

- bytes and bytearray are used for manipulating binary data.
- The Bytes type in Python is immutable and stores a sequence of values ranging from 0-255 (8-bits). You can get the value of a single byte by using an index like an array, but the values can not be modified.
- It is defined like a string with a ‘b’ as its prefix

```
x = b’character’
print(len(x))
print(x)
print(x[2])
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- The Bytearray Type is used to create a mutable object you need to use the bytearray type. With a bytearray you can do everything you can with other mutables like push, pop, insert, append, delete, and sort.

```
y = bytearray(5)
print(y)
```

- memoryview is used for accessing memory of other binary objects.

```
print(memoryview(y))
```

OPERATORS IN PYTHON

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

CNT...

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y

CNT...

*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

CNT...

//	Floor division	x // y

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5

CNT...

+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3

CNT...

%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

CNT...

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y

CNT...

>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

CNT...

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10

CNT...

or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

CNT...

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y

CNT...

is not	Returns True if both variables are not the same object	x is not y

CNT...

Python Membership Operators

Membership operators are used to test if a sequence is present in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y

CNT...

not in	Returns True if a sequence with the specified value is not present in the object	x not in y

CNT...

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1

CNT...

^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off to the next place
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

STRING FUNCTIONS

Python has a set of built-in methods that you can use on strings.

All string methods returns new values. They do not change the original string.

- **count()**

  returns the number of times a specified value appears in the string

  Eg:

  string = ‘celebrate’

  print(string.count(‘e’))

  print(string.count(‘e’, 4)) # start index to search; default 0

  print(string.count(‘e’, 4 ,7)) #start and end index; default end of the string

CONT...

- **center()**

  center align the string, using a specified character (space is default) as the fill character

  Eg:

  string = ‘Python’

  print(string.center(15,’*’))

  Output: *****Python****

CONT...

- **encode()**

returns an encoded version of the string

Eg:

```
txt = "My name is Sam"
x = txt.encode() # default : encoding = ‘UTF-8’/UTF-16, UTF-32
print(x)
Output: b'My name is Sam'
```

CONT...

- **endswith()**

  returns True if the string ends with the specified value, otherwise False

  Eg:

  ```
  txt = "My name is Sam“
  print(txt.endswith('m'))
  print(txt.endswith('m',1,5))  # (start index, end index)
  ```

- **startswith()**

  returns True if the string starts with the specified value

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

Eg:

```
txt = "My name is Sam“
print(txt.startswith(‘M'))
print(txt.startswith('m',5)) # start index
```

- **swapcase()**

  Make the lower case letters upper case and the upper case letters lower case

  Eg:print('WindoWs'.swapcase())

CONT...

- **casefold()**

  converts string into lower case; for caseless matching of unicode characters

  Eg:

  Print(“Hello World”.casefold())

  Output: hello world

- **capitalize()**

  converts the first character to upper case

  Eg:

  string = ‘learn’

  string.capitalize()

CONT...

- **expandtabs()**

  sets the tab size to the specified number of whitespaces

  Eg:

```
txt = "H\te\tl\tl\to"
x =  txt.expandtabs(4)
print(x)
```

- **format()**

  formats the specified value(s) and insert them inside the string's placeholder; placeholder is defined using curly brackets {}

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

- **format_map()**

used to return the value of a dictionary key

Eg:

```
profession = {'name': ['Barry', 'Bruce'],
              'profession': ['Engineer', 'Doctor'],
              'age': [30, 31]}
print('{name[0} is an {profession[0]} and he is {age[0} years old.'.format_map(profession))
```

CONT...

- **find()**

  finds the first occurrence of the specified value, returns -1 if not found

  Eg:

  ```
  txt = "Hello, welcome to my world."
  print(txt.find("welcome"))
  print(txt.find("welcome“, 3, 8))
  ```

- **rfind()**

  finds the last occurrence of the specified value; returns -1 if not found

CONT...

- **index()**

  finds the first occurrence of the specified value; same as find but raises error when not find.

- **rindex()**

  finds the last occurrence of the specified value; raises error when not found

CONT...

- **isalnum()**

  check if all the characters in the text is alphanumeric

  Eg:

  txt = "Company 12“

  txt1 = "Company12"

  print(txt.isalnum())

  print(txt1.isalnum())

- **isalpha()**

returns True if all the characters are alphabet letters

CONT...

- **isnumeric()**

  returns True if all the characters are numeric (0-9, subscripts etc,

  Eg:

```
a = "\u0030" #unicode for 0
b = "\u00B2" #unicode for &sup2;
c = "10km2"
d = "-1"
print(a.isnumeric())
print(b.isnumeric())
print(c.isnumeric())
print(d.isnumeric())
```

CONT...

- **isdigit()**

  returns True if all the characters are digits, otherwise False; supports subscripts, superscripts etc.

  Eg:

  a = "\u0030" #unicode for 0

  b = "\u00B2" #unicode for ²

  print(a.isdigit())

  print(b.isdigit())

  Output: True

  True

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

- **islower()**

  check if all the characters in the text are in lower case

- **isupper()**

  check if all the characters in the text are in upper case

- **isprintable()**

  returns True if all the characters are printable

  Eg:

  print('hello\n'.isprintable())

- **isspace()**

  check if all the characters in the text are whitespaces

CONT...

- **isidentifier()**

  check if the string is a valid identifier

- **istitle()**

  check if each word start with an upper case letter

  Eg:

  print('Check Whether This Is A Title'.istitle())

CONT...

- **join()**

  method takes all items in an iterable and joins them into one string; a string must be specified as the separator

  Eg:

  ```
  mySeparator = " TEST "
  mytup= ('a','b','c')
  print(mySeparator.join(mytup))
  ```

CONT...

- **ljust()**

  left align the string, using a specified character (space is default) as the fill character

  Eg:

  txt = 'Simple'

  print(txt.ljust(10,'*'))

  Output: Simple****

- **rjust()**

  right align the string, using a specified character (space is default) as the fill character

  Eg:

  txt = 'Simple'

  print(txt.rjust(10,'*'))

  Output: ****Simple

CONT...

- **lower()**

  converts every character in the string to lower case

- **upper()**

  converts every character in the string to upper case

- **rstrip()**

  method removes any trailing characters

  Eg:

  print(',,,Hi...how...are...you,,,,'.rstrip(','))

  Output: ,,,Hi...how...are...you

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

- **lstrip()**

  method removes any leading characters

  Eg:

  print(',,,Hi...how...are...you,,,,'.lstrip(','))

  Output: Hi...how...are...you,,,,

- **strip()**

  removes any leading and trailing characters (space is the default leading character to remove)

  Eg:

  print(',,,Hi...how...are...you,,,,'.strip(','))

  Output: Hi...how...are...you

CONT...

- **partition()**

  searches for a specified string, and splits the string into a tuple containing three elements

  Eg:

```
txt = "This is a partition statement"
print(txt.partition('is')) # checks for the first occurrence
print(txt.partition('k')) # 3 parts; 2 of which will be empty
```

CONT...

- **rpartition()**

  searches for the last occurrence of a specified string, and splits the string into a tuple containing three elements

- **replace()**

  replaces a specified phrase with another specified phrase

  Eg:

```
txt = "This is a replace statemnet"
print(txt.replace('is','as')) # default replace all occurences
print(txt.replace('is','as',1)) # only replace 1 occurence
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

- **maketrans()**

  returns a mapping table that can be used with the translate() method to replace specified characters

  Eg:

```
txt = "Hello Sam!“
#specify strings to be replaced
mytable1 = txt.maketrans("Sal", "Pet")
#Replacement by dictionary
mytable2 = txt.maketrans({'S':'P', 'a':'e'})
```

CONT...

```
#Replacement and removal
mytable3 = txt.maketrans("Sal", "Pet","H")
print(mytable1)
print(mytable2)
print(mytable3)
```

- **translate()**

  Returns the translated string

  Eg:

  ```
  print(txt.translate(mytable1))
  print(txt.translate(mytable2))
  print(txt.translate(mytable3))
  ```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CONT...

- **split()**

splits the string at the specified separator, and returns a list

Eg:

```
txt = "Hello, I am Peter, I am 26 years old"
print(txt.split(", ")) # default split value -1 i.e., all
print(txt.split(',',1)) # splits at only one place
```

CONT...

- **splitlines()**

  splits a string into a list where splitting is done at line breaks

  Eg:

  ```
  txt = "Hello, I am Peter\nI am 26 years old"
  print(txt.splitlines())
  ```

- **title()**

  converts the first character of each word to upper case

  Eg:

  ```
  txt = "Hello, I am Peter"
  print(txt.title())
  ```

CONT...

- **zfill()**

  adds zeros at the beginning of the string, until it reaches the specified length

  Eg:

  ```
  txt = "Hello, I am Peter"
  print(txt.zfill(25))
  ```

MATH FUNCTIONS

In python a number of mathematical operations can be performed with ease by importing a module named “math” which defines various functions which makes our tasks easier.

- ceil() :- This function returns the smallest integral value greater than the number. If number is already integer, same number is returned.
- floor() :- This function returns the greatest integral value smaller than the number. If number is already integer, same number is returned.
- fabs() :- This function returns the absolute value of the number.

CNT...

- factorial() :- This function returns the factorial of the number. An error message is displayed if number is not integral.
- copysign(a, b) :- This function returns the number with the value of ‘a’ but with the sign of ‘b’. The returned value is float type.
- gcd() :- This function is used to compute the greatest common divisor of 2 numbers mentioned in its arguments. This function works in python 3.5 and above.

CNT...

- import math

  a = 2.3

  print ("The ceil of 2.3 is : ", end="")# returning the ceil of 2.3

  print (math.ceil(a))

  Output:The ceil of 2.3 is : 3

  print ("The floor of 2.3 is : ", end="")# returning the floor of 2.3

  print (math.floor(a))

  Output:The floor of 2.3 is : 2

CNT...

- import math

  a = -10

  b= 5

  print ("The absolute value of -10 is : ", end="")# returning the absolute value.

  print (math.fabs(a))

  Output:The absolute value of -10 is : 10.0

CNT...

print ("The factorial of 5 is : ", end="")# returning the factorial of 5

print (math.factorial(b))

Output:The factorial of 5 is : 120

- import math

  a = -10

  b = 5.5

  c = 15

CNT...

d = 5

print ("The copy signed value of -10 and 5.5 is : ", end="")# returning the copy signed value.

print (math.copysign(5.5, -10))

print ("The gcd of 5 and 15 is : ", end="")# returning the gcd of 15 and 5

print (math.gcd(5,15))

Output:The copy signed value of -10 and 5.5 is : -5.5

The gcd of 5 and 15 is : 5

USER INPUT IN PYTHON

Most programs today use a dialog box as a way of asking the user to provide some type of input. While Python provides us with two inbuilt functions to read the input from the keyboard.

- raw_input (prompt)
- input (prompt)

CNT...

raw_input( ) : This function works in older version (like Python 2.x). This function takes exactly what is typed from the keyboard, convert it to string and then return it to the variable in which we want to store.

```
g = raw_input("Enter your name : ")

print g
```

Output:Enter your name : python

python

CNT...

input( ) : This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python.

```
val = input("Enter your value: ")

print(val)
```

Output:Enter your value:123

123

DECISION MAKING STATEMENTS

Decision making statements in programming languages decides the direction of flow of program execution. Decision making statements available in python are:

- if statement:

  if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

CNT...

Syntax:

```
if condition:
# Statements to execute
```

```
eg:a = 33
b = 200
if b > a:
        print("b is greater than a")
```

CNT...

- elif statement:

  The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

  if condition:

    statments

  elif condition:

    statements

CNT...

eg: a = 33

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

CNT...

- if else:

  The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

CNT...

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
     # condition is false
```

CNT...

```
i = 20;

if (i < 15):

    print ("i is smaller than 15")

else:

    print ("i is greater than 15")

print ("i'm not in if and not in else Block")
```

CNT...

- Nested-if:

A nested if is an if statement that is the target of another if statement. Nested if statements means an if statement inside another if statement.Python allows us to nested if statements within if statements. i.e, we can place an if statement inside another if statement

CNT...

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
 # if Block is end here
```

CNT...

```
eg:i = 10
    if (i == 10):
        if (i < 15):
            print ("i is smaller than 15")
    if (i < 12):
       print ("i is smaller than 12 too")
    else:
       print ("i is greater than 15")
```

LOOPING STATEMENTS

- for loop

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

Syntax:

```
for iterating_variable in sequence:
```

CNT...

eg:fruits = ["Apple", "Banana", "Cherry"]

```
for x in fruits:
        print(x)
```

Output:

Apple

Banana

Cherry

CNT...

- <u>while loop:</u>

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

  Syntax:

  While condition:

  statement

CNT...

eg:i = 1

```
    while i < 4:
    print(i)
    i += 1
```

Output:1

2

3

LOOP CONTROL STATEMENTS

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements:

- Continue Statement:

  It returns the control to the beginning of the loop.With the continue statement we can stop the current iteration of the loop, and continue with the next.

CNT...

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

```
  for x in fruits:
  if x == "banana":
    continue
  print(x)
```

Output:

```
    apple
    cherry
```

CNT...

- Break Statement

With the break statement we can stop the loop before it has looped through all the items.

eg:fruits = ["apple", "banana", "cherry"] Output:apple

```
for x in fruits:                                    banana
    print(x)
    if x == "banana":
      break
```

CNT...

- Pass statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.The pass statement is a *null* operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet .

CNT...

```
eg:a = 33
    b = 200
    if b > a:
        pass
```

having an empty if statement like this, would raise an error without the pass statement

FUNCTION IN PYTHON

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

  In Python a function is defined using the def keyword:

eg:

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

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- Calling a Function

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

  eg:

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

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.

Arguments are often shortened to *args* in Python documentations.

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective a parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that are sent to the function when it is called.

CNT...

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:

```
eg:def my_function(fname):
        print(fname + " Refsnes")

   my_function("Emil")

   my_function("Tobias")

   my_function("Linus")
```

Output: Emil Refsnes

Tobias Refsnes

Linus Refsnes

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

eg:def my_function(fname, lname):

```
        print(fname + " " + lname)

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

Output:Emil Refsnes

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Arbitrary Arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a *tuple* of arguments, and can access the items accordingly.

Arbitrary Arguments are often shortened to **args* in Python documentations.

CNT...

Eg:

```
def my_function(*kids):
        print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

Output:The youngest child is Linus

CNT...

Keyword Arguments

You can also send arguments with the *key* = *value* syntax.

This way the order of the arguments does not matter.

eg:

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

Output: The youngest child is Linus

CNT...

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a *dictionary* of arguments, and can access the items accordingly.

CNT...

```
eg:def my_function(**kid):
        print("His last name is " + kid["lname"])
    my_function(fname = "Tobias", lname = "Refsnes")
```

Output:His last name is Refsnes

CNT..

Default Parameter Value

If we call the function without argument, it uses the default value.

eg:def my_function(country = "Norway"):

```
    print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

CNT...

Output:I am from Sweden

I am from India

I am from Norway

I am from Brazil

CNT...

Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function

CNT...

eg:

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

Output:apple
banana
cherry

CNT...

Return Values

To let a function return a value, use the return statement:

Eg:

```
def my_function(x):
  return 5 * x
print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:15

25

45

TECHOLAS

TECHNOLOGY DEMYSTIFIED

CNT...

Recursion

Python accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

CNT...

```
def tri_recursion(k):
  if(k > 0):
        result = k + tri_recursion(k - 1)
        print(result)
  else:
        result = 0
 return result
print("\n\nRecursion Example Results")
```

CNT...

tri_recursion(6)

Output:Recursion Example Results

1

3

6

10

15

21

CNT....

In this example, tri_recursion() is a function that we have defined to call itself ("recurse"). We use the k variable as the data, which decrements (-1) every time we recurse. The recursion ends when the condition is not greater than 0 (i.e. when it is 0).

GLOBAL KEYWORD IN PYTHON

- Global keyword is a keyword that allows a user to modify a variable outside of the current scope.
- It is used to create global variables from a non-global scope i.e inside a function.
- Global keyword is used inside a function only when we want to do assignments or when we want to change a variable.
- Global is not needed for printing and accessing.
- To access a global variable inside a function there is no need to use global keyword.

CNT...

Rules of global keyword:

- If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.
- Variables that are only referenced inside a function are implicitly global.
- We Use global keyword to use a global variable inside a function.
- There is no need to use global keyword outside a function

CNT...

```
eg:a = 15# global variable
    b = 10# global variable
    def add():# function to perform addition
        c = a + b
        print(c)
   add()# calling a function
```

Output:25

ANONYMOUS FUNCTION

In Python, anonymous function is a function that is defined without a name.

While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

Syntax of Lambda Function in python:

```
lambda arguments: expression
```

CNT...

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

eg:double = lambda x: x * 2

```
    print(double(5))
```

Output: 10

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

In the program, lambda x: x * 2 is the lambda function. Here x is the argument and x * 2 is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier double.

We use lambda functions when we require a nameless function for a short period of time.

CNT...

Lambda functions are used along with built-in functions like filter(), map() and reduce().

filter():

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.

CNT...

eg:my_list = [1, 5, 4, 6, 8, 11, 3, 12]

```
new_list = list(filter(lambda x: (x%2 == 0) , my_list))

print(new_list)
```

Output: [4, 6, 8, 12]

CNT...

map():

The map() function in Python takes in a function and a list.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

eg:my_list = [1, 5, 4, 6, 8, 11, 3, 12] Output: [2, 10, 8, 12, 16, 22, 6, 24]

```
new_list = list(map(lambda x: x * 2 , my_list))

print(new_list)
```

CNT...

Reduce:

The reduce() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list. This is a part of functools module.

Eg: from functools import reduce Output:193

```
li = [5, 8, 10, 20, 50, 100]

sum = reduce((lambda x, y: x + y), li)

print (sum)
```

MODULES IN PYTHON

Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a *module.*

Definitions from a module can be *imported* into other modules.

A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended. Within a module, the module's name (as a string) is available as the value of the global variable __name__.

CNT...

Create a file called fibo.py in the current directory with the following contents:

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()
```

CNT....

```
def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
        return result
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Now enter the Python interpreter and import this module with the following command:

```
>>> import fibo
```

Using the module name you can access the functions:

```
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

CNT...

If you intend to use a function often you can assign it to a local name:

```
>>> fib = fibo.fib
>>> fib(500)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

CNT...

- A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module name is encountered in an import statement.
- They are also run if the file is executed as a script.
- Each module has its own private symbol table, which is used as the global symbol table by all functions defined in the module.

CNT...

- Thus, the author of a module can use global variables in the module without worrying about accidental clashes with a user's global variables.
- On the other hand, if you know what you are doing you can touch a module's global variables with the same notation used to refer to its functions, modname.itemname.
- Modules can import other modules. It is customary but not required to place all import statements at the beginning of a module (or script, for that matter).

CNT...

- The imported module names are placed in the importing module's global symbol table.
- There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

Eg: >>> from fibo import fib, fib2

>>> fib(500)

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

CNT...

- There is even a variant to import all names that a module defines:

```
>>> from fibo import *
>>> fib(500)
    0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This imports all names except those beginning with an underscore (_)

CNT...

- If the module name is followed by as, then the name following as is bound directly to the imported module.

```
>>> import fibo as fib
>>> fib.fib(500)
    0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

This is effectively importing the module in the same way that import fibo will do, with the only difference of it being available as fib.

CNT...

- It can also be used when utilising “from” with similar effects:

```
>>> from fibo import fib as fibonacci
>>> fibonacci(500)
    0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

CNT...

Executing modules as scripts

When you run a Python module with:

python fibo.py <arguments>

the code in the module will be executed, just as if you imported it, but with the __name__ set to "__main__".

CNT...

That means that by adding this code at the end of your module:

```
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

You can make the file usable as a script as well as an importable module, because the code that parses the command line only runs if the module is executed as the “main” file:

CNT...

$ python fibo.py 50

0 1 1 2 3 5 8 13 21 34

If the module is imported, the code is not run:

>>> import fibo

>>>

CNT...

The Module Search Path

When a module named spam is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named spam.py in a list of directories given by the variable sys.path. sys.path is initialized from these locations:

- The directory containing the input script (or the current directory when no file is specified).
- PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH).

CNT...

After initialization, Python programs can modify sys.path.

The directory containing the script being run is placed at the beginning of the search path, ahead of the standard library path. This means that scripts in that directory will be loaded instead of modules of the same name in the library directory.

This is an error unless the replacement is intended.

CNT...

Compiled Python files

- To speed up loading modules, Python caches the compiled version of each module in the __pycache__ directory under the name module.*version*.pyc, where the version encodes the format of the compiled file; it generally contains the Python version number.
- For example, in CPython release 3.3 the compiled version of spam.py would be cached as __pycache__/spam.cpython-33.pyc. This naming convention allows compiled modules from different releases and different versions of Python to coexist.

CNT...

Standard Modules

- Python's standard library is very extensive, offering a wide range of functionalities.
- Python 3's standard modules such as the statistics module, the math module and the random module.

CNT...

Statistics Module

This module provides functions for calculating mathematical statistics of numeric (Real-valued) data.

Some of the most commonly used functions of the module are:

- **mean()**: Arithmetic mean ('average') of data.

```
import statistics
A = [5,10,6,21,5,17,14,8,3,9]
mean = statistics.mean(A)
print('The mean of A is ', mean)
```

Output: The mean of A is 9.8

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **median()**: Median (middle value) of data.

```
import statistics
B = [1,7,13,24,35,37,42,44,53,55]
median = statistics.median(B)
print('Median of B is:', median)
```

Output: Median of B is: 36.0

CNT...

- **median_low()**: Low median of the data is the lower of two middle values when the number of data elements is even, else it is the middle value.

```
import statistics
B = [1,7,13,24,35,37,42,44,53,55]
low_median = statistics.median_low(B)
print('Low Median of B is:', low_median)
```

Output: Low Median of B is: 35

CNT...

- **median_high()**: High median of the data is the larger of two middle values when the number of data elements is even, else it is the middle value.

```
import statistics
B = [1,7,13,24,35,37,42,44,53,55]
high_median = statistics.median_high(B)
print('High Median of B is:', high_median)
```

Output: High Median of B is: 37

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **mode()**: Mode (most common value) of discrete data.

```
import statistics
C = [1,1,2,2,2,2,4,4]
most_common_item = statistics.mode(C)
print('The most common item of C is:', most_common_item)
```

Output: The most common item of C is: 2

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT....

- **variance()**: Return the sample variance of data, an iterable of at least two real-valued numbers. Variance measures the spread of data. A low variance indicates that all the data is clustered around the mean and a high variance indicates that the data is spread out.

```
import statistics
A = [5,10,6,21,13,17,14,8,3,9]
variance = statistics.variance(A)
print('The variance of A is:', variance)
```

Output: The variance of A is: 31.822222222222226

CNT...

- **stdev()**: Return the sample standard deviation (the square root of the sample variance).

```
import statistics
A = [5,10,6,21,13,17,14,8,3,9]
std = statistics.stdev(A)
print('The std of A is:', std)
```

Output: The std of A is: 5.641118880348315

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Math Module

This module provides access to the mathematical functions defined by the C standard.

Some of the most commonly used functions of the module are:

- **sqrt(x):** Return the square root of x.

```
import math
x = 81
print('The square root of', x, 'is', math.sqrt(x))
```

Output: The square root of 81 is 9.0

CNT...

- **ceil(x):** Return the ceiling of x, the smallest integer greater than or equal to x.

```
import math
x = 5.6
print('The ceil of', x, 'is', math.ceil(x))
```

Output: The ceil of 5.6 is 6

CNT...

- **floor(x):** Return the floor of x, the largest integer less than or equal to x.

```
import math
x = 5.6
print('The floor of', x, 'is', math.floor(x))
```

Output: The floor of 5.6 is 5

CNT....

- **factorial(x):** Return x factorial.

```
import math
x = 6
print('The factorial of', x, 'is:', math.factorial(x))
```

Output: The factorial of 6 is: 720

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **exp(x):** Return e raised to the power x, where e = 2.718281... is the base of natural logarithms.

```
import math
x = 2
print('e ^', x,'=', math.exp(x))
```

Output: e ^ 2 = 7.38905609893065

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **pow(x,y):** Return x raised to the power y

```
import math
base = 2; x = 4;
print(base, '^', x, '=', math.pow(base,x))
```

Output: 2 ^ 4 = 16.0

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **log(x, base):** With one argument, return the natural logarithm of x (to base e). With two arguments, return the logarithm of x to the given base.

```
import math
base = 2; x = 16;
print('log of', x, 'to the base of', base, '=', math.log(x, base))
```

Output: log of 16 to the base of 2 = 4.0

CNT...

- **pi:** This is actually a the mathematical constant pi = 3.141592..., to available precision.

```
import math
print('Pi value:', math.pi)
```

Output: Pi value: 3.141592653589793

CNT...

- **e:** This is also the mathematical constant e = 2.718281..., to available precision.

```
import math
print('e value:', math.e)
```

Output: e value: 2.718281828459045

CNT...

- **radians(x):** Convert angle x from degrees to radians.

```
import math
print('Radians of 90 degrees:', math.radians(90))
```

Output: Radians of 90 degrees: 1.5707963267948966

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **degrees(x):** Convert angle x from radians to degrees.

```
import math
print('Degrees of pi/2:', math.degrees(math.pi/2))
```

Output: Degrees of pi/2: 90.0

CNT...

Random module

This module implements pseudo-random number generators for various distributions.

Some of the most commonly used functions of the module are:

- **randint(a, b):** Return a random integer x, which belongs to the range [a,b].

```
import random
result = random.randint(1, 11)
print('The random integer number in range [1,11] is:', result)
```

Output: The random integer number in range [1,11] is: 10

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **shuffle()**:Shuffle the sequence x in place.

  import random

  X=list(range(1,11))

  random.shuffle(X)

  print('Shuffled list:',X)

  Output:Shuffled list:[6,9,2,8,5,10,7,3,4,1]

CNT...

- **random.randrange()**: Returns a randomly selected element from the range created by the start, stop and step arguments. The value of start is 0 by default.

  import random

  random.randrange(1,10)

  output:2

  random.randrange(1,10,2)

  output:5

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

- **choice(seq):** Return a random element from the non-empty sequence seq.

```
import random
coin_side = random.choice(['H','T'])
print('Flip a coin simulation using choice function. Result:', coin_side)
```

Output: Flip a coin simulation using choice function. Result: T

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

random.randrange(0,101,10)

Output:80

Other modules are:

datetime module

- The datetime module supplies classes for manipulating dates and times in both simple and complex ways.

CNT....

- While date and time arithmetic is supported, the focus of the implementation is on efficient member extraction for output formatting and manipulation.
- The module also supports objects that are timezone aware.

```
import datetime

x = datetime.datetime.now()

print(x)
```

Output:2020-03-19 21:23:18.801745

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

The datetime module has many methods to return information about the date object.

```
import datetime

x = datetime.datetime.now()

print(x.year)

print(x.strftime("%A"))
```

Output: 020

Thursday

CNT...

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

Output:2020-05-17 00:00:00

CNT...

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

strftime() Method

- The datetime object has a method for formatting date objects into readable strings.
- The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

CNT...

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

Output:June

CNT....

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday

CNT...

%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December

CNT...

%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17

CNT....

%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08

CNT...

%f	Microsecond 000000-999999	548513
%j	Day number of year 001-366	365

CNT...

%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018

CNT...

%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

CNT...

Packages

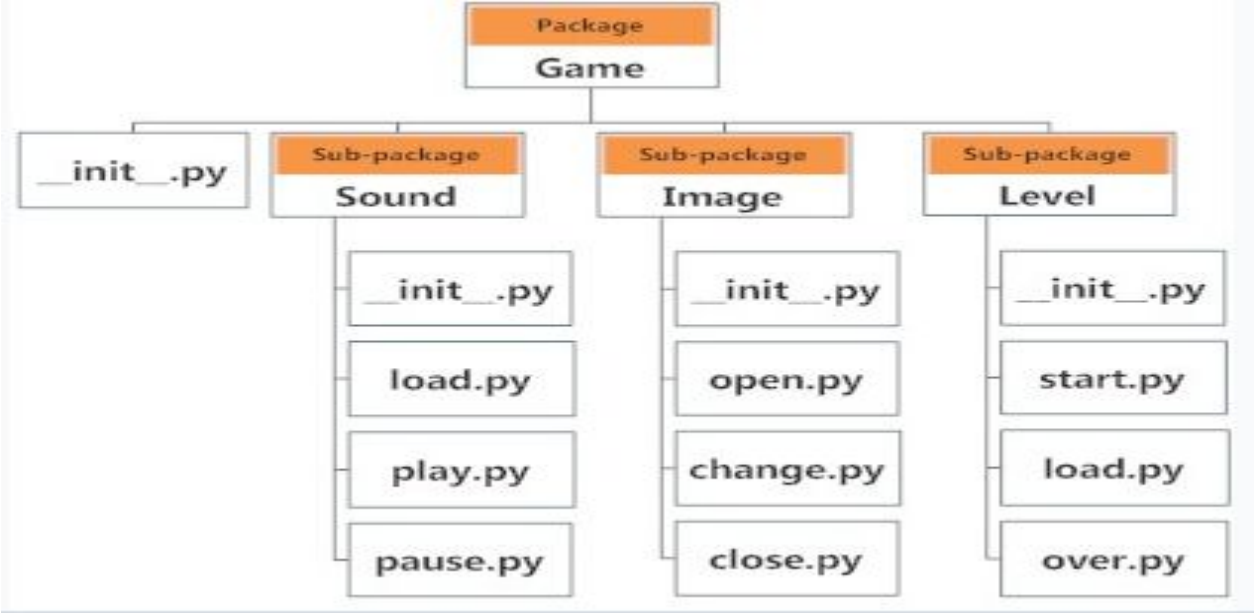
Python has packages for directories and modules for files.

As a directory can contain sub-directories and files, a Python package can have sub-packages and modules.

A directory must contain a file named __init__.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

CNT...

Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.



TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

Importing module from a package

We can import modules from packages using the dot (.) operator.

For example, if want to import the start module in the above example, it is done as follows.

import Game.Level.start

CNT...

Now if this module contains a function named select_difficulty(), we must use the full name to reference it.

Game.Level.start.select_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows.

from Game.Level import start

CNT...

We can now call the function simply as follows.

start.select_difficulty(2)

Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.

from Game.Level.start import select_difficulty

Now we can directly call this function.

select_difficulty(2)

CNT...

Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding.

While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.

OOP

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

- attributes
- behavior

CNT...

Parrot is an object,

- name, age, color are attributes
- singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

CNT...

In Python, the concept of OOP follows some basic principles:

Inheritance	A process of using details from a new class without modifying existing class.
Encapsulation	Hiding the private details of a class from other objects.
Polymorphism	A concept of using common operation in different ways for different data input.

CNT...

Class

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

CNT...

The example for class of parrot can be :

```
class Parrot:

    pass
```

Here, we use class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

CNT...

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is object of class Parrot.

CNT...

Creating Class and Object in Python

```
class Parrot:

    species = "bird"# class attribute

    def __init__(self, name, age):  # instance attribute

        self.name = name

        self.age = age

blu = Parrot("Blu", 10)# instantiate the Parrot class
```

CNT...

```
woo = Parrot("Woo", 15)

print("Blu is a {}".format(blu.__class__.species))# access the class attributes

print("Woo is also a {}".format(woo.__class__.species))

print("{} is {} years old".format( blu.name, blu.age))# access the instance attributes

print("{} is {} years old".format( woo.name, woo.age))
```

CNT...

output :

Blu is a bird

Woo is also a bird

Blu is 10 years old

Woo is 15 years old

In the program, we create a class with name Parrot. Then, we define attributes. The attributes are a characteristic of an object.

CNT...

Then, we create instances of the Parrot class. Here, blu and woo are references (value) to our new objects.

Then, we access the class attribute using __class __.species. Class attributes are same for all instances of a class.

Similarly, we access the instance attributes using blu.name and blu.age.

However, instance attributes are different for every instance of a class.

TECHOLAS

TECHNOLOGY DEMYSTIFIED

CNT...

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Creating Methods in Python

```
class Parrot:

    def __init__(self, name, age):# instance attributes

        self.name = name

        self.age = age
```

CNT...

```
    def sing(self, song):# instance method

        return "{} sings {}".format(self.name, song)

    def dance(self):

        return "{} is now dancing".format(self.name)

blu = Parrot("Blu", 10)# instantiate the object

print(blu.sing("'Happy'"))# call our instance methods

print(blu.dance())
```

CNT...

output:

Blu sings 'Happy'

Blu is now dancing

In the above program, we define two methods i.e sing() and dance(). These are called instance method because they are called on an instance object i.e blu.

CNT...

Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

```
class Bird:# parent class

    def __init__(self):

        print("Bird is ready")
```

CNT...

```
        def whoisThis(self):

                print("Bird")

        def swim(self):

                print("Swim faster")

class Penguin(Bird):# child class

    def __init__(self):

                super().__init__()# call super() function
```

CNT....

```
            print("Penguin is ready")
    def whoisThis(self):
        print("Penguin")
    def run(self):
        print("Run faster")
peggy = Penguin()
```

CNT...

peggy.whoisThis()

peggy.swim()

peggy.run()

output:

Bird is ready

Penguin is ready

Penguin

CNT...

Swim faster

Run faster

In the program, we created two classes i.e. Bird (parent class) and Penguin (child class).

The child class inherits the functions of parent class. We can see this from swim() method.

CNT...

Again, the child class modified the behavior of parent class. We can see this from whoisThis() method.

Furthermore, we extend the functions of parent class, by creating a new run() method.

Additionally, we use super() function before __init__() method.

This is because we want to pull the content of __init__() method

from the parent class into the child class.

CNT...

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation.

In Python, we denote private attribute using underscore as prefix i.e single “ _ “ or double “ __“.

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

```
class Computer:

    def __init__(self):

        self.__maxprice = 900

    def sell(self):

        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):

        self.__maxprice = price
```

CNT...

```
c = Computer()

c.sell()

c.__maxprice = 1000# change the price

c.sell()

c.setMaxPrice(1000)# using setter function

c.sell()
```

CNT...

output:

Selling Price: 900

Selling Price: 900

Selling Price: 1000

In the above program, we defined a class Computer. We use __init__() method to store the maximum selling price of computer.

We tried to modify the price.

CNT...

However, we can’t change it because Python treats the __maxprice as private attributes. To change the value, we used a setter function i.e setMaxPrice() which takes price as parameter.

CNT...

Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

CNT...

```
class Parrot:

    def fly(self):

        print("Parrot can fly")

    def swim(self):

        print("Parrot can't swim")

class Penguin:

    def fly(self):

        print("Penguin can't fly")
```

CNT...

```
        def swim(self):
                print("Penguin can swim")
def flying_test(bird):# common interface
    bird.fly()
blu = Parrot()#instantiate objects
peggy = Penguin()
flying_test(blu)# passing the object
flying_test(peggy)
```

CNT...

output:

Parrot can fly

Penguin can't fly

In the program, we defined two classes Parrot and Penguin. Each of them have common method fly() method. However, their functions are different.

To allow polymorphism, we created common interface i.e flying_test() function that can take any object. Then, we passed the objects blu and peggy in the flying_test() function, it ran effectively.

CNT...

Constructors in Python

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

__init__() function is a special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

CNT....

```
class ComplexNumber:
    def __init__(self,r = 0,i = 0):
        self.real = r
        self.imag = i
    def getData(self):
        print("{0}+{1}j".format(self.real,self.imag))
c1 = ComplexNumber(2,3)# Create a new ComplexNumber object
c1.getData()# Call getData() function
```

CNT...

c2 = ComplexNumber(5)# Create another ComplexNumber object

c2.attr = 10# and create a new attribute 'attr'

print((c2.real, c2.imag, c2.attr))

C1.attr

Output:

2+3j

(5, 0, 10)

AttributeError: 'ComplexNumber' object has no attribute 'attr

CNT...

In the example, we define a new class to represent complex numbers. It has two functions, __init__() to initialize the variables (defaults to zero) and getData() to display the number properly.

Attributes of an object can be created on the fly. We created a new attribute attr for object c2 and we read it as well.

But this did not create that attribute for object c1.

CNT...

Operator overloading in Python

Operator Overloading means giving extended meaning beyond their predefined operational meaning.

For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class.

The same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading.

CNT...

```
print(1 + 2)#addition of two numbers

print("Geeks"+"For") #concatenation of two strings

print(3 * 4)#multiplication of two numbers

print("Geeks"*4)#repetition of a string
```

Output:

```
3

GeeksFor
```

CNT...

12

GeeksGeeksGeeksGeeks

Eg:

```
class A:
        def __init__(self, a):
                self.a = a
```

CNT...

```
def __add__(self, o): # adding two objects

        return self.a + o.a

ob1 = A(1)

ob2 = A(2)

ob3 = A("Geeks")

ob4 = A("For")

print(ob1 + ob2)

print(ob3 + ob4)
```

CNT...

Output :

3

GeeksFor

CNT...

Method overloading

In python, we can define a method in such a way that there are multiple ways to call it. Given a single method or function,we can specify the number of parameters ourselves. Depending on the function definition,it can be called with zero,one,or two more parameters.This is known as method overloading

CNT..

```
Class Human:
    Def sayHello(self,name=None):
        if name is not None:
            print(“Hello”+name)
        else:
            print(“Hello”)
obj=Human()
obj.sayHello()
```

CNT...

obj.sayHello(‘Guido)

Output:

Hello

Hello Guido

CNT...

Method overriding

Method overriding is an ability of any object-oriented programming language that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

TECHOLAS
TECHNOLOGY DEMYSTIFIED

CNT...

```
class Parent():
    def __init__(self):
        self.value = "Inside Parent"
    def show(self):  # Parent's show method
        print(self.value)
class Child(Parent):# Defining child class
        def __init__(self):
```

CNT...

```
        self.value = "Inside Child"
    def show(self): # Child's show method
        print(self.value)
obj1 = Parent()# Driver's code
obj2 = Child()
obj1.show()
obj2.show()
```

CNT...

Output:

Inside Parent

Inside Child

EXCEPTION HANDLING

Errors in python can be of 2 types: **syntax errors** and **exceptions**.

Syntax errors are due to wrong syntax which leads to the termination of the code

Eg:

```
x = 10
if x<0
    print('The value is negative')
```

Here, after the if clause, ':' is not used. This is a syntax error.

CNT...

Exceptions are raised when the code is syntactically correct, but the code result in an error. It changes the normal flow of the program.

Eg:

```
x = 5
print(x/0)
```

This leads to an exception since division by zero is not defined. But this statement is syntactically correct.

Catching exceptions- try and except statements

Statements that can cause an exception are placed inside a try block. If it raise an exception, it will be thrown from the try block which is caught by the except block.

The code to handle the exception is placed inside the except block.

CNT...

Eg:

```
a = [2,8,10]
try:
   print('First element:', a[0])
   print('Fourth element:', a[3]) # raises exception
except:
   print('The index is out of range') #handles the exception
```

Catching a specific exception

A try block can have more than one except clause for handling different exceptions. Only one exception handler will be executed at a time.

Syntax

```
try:
   statements(s)
except IndexError:
   statement(s)
except ValueError:
   statements(s)
```

CNT...

```
def func(a):
   If a<4:
        b = a/(a-3)
   print('Value of b:', b)
try:
   func(2)
   func(3)
   func(5)
except ZeroDivisionError:
   print('Zero division error occurred')
except NameError:
   print('Name error occurred')
```

try...else

An else clause can be used on the try-except block after all the except clauses.

The code will enter the else block only if the try block doesn't raise an exception.

Eg:

```
def arithmetic(a,b):
   try:
      c = (a+b)/(a-b)
   except ZeroDivisionError:
      print('Zero division error')
   else:
      print(c)
arithmetic(9,2)
arithmetic(4,3)
```

try..except..else..finally

The finally clause is always executed after the normal termination of the try block or after the try block terminates due to some exception.

```
try:
   statements(s)
except:
   executed if exception raised
else:
   execute if no exception
finally:
   always executed
```

Raising an exception

The ‘raise’ statement allows the programmer to force a specific exception to occur. This will stop the execution of the program.

Eg:1

```
x = -1
if x<0:
   raise Exception (‘Exception: number below zero’)
```

Eg:2

```
a = ‘abc’
if type(a) is not int:
   raise Exception (‘Exception: Not an integer’)
```

Argument of an exception

We can pass arguments to an exception for cases when there is a need for additional information from an exception raised by Python.

Eg:

```
string = ‘parameter’
try:
   b = string/100
except Exception as arg:
   print(‘This is the argument :’, arg)
```

User defined exception class

Users can define their own exceptions using the Exception class in python. All the exceptions need to be derived from the Exception class.

Eg:

```
class Custom_Exception:
   pass
raise Custom_Exception('This is a custom exception')
```

CONT...

Eg:

```
class MyError(Exception):
   def __init__(self, value):
      self.value = value
try:
   raise(MyError(7))
except MyError as error:
   print('A new exception has occurred', error.value)
```

Standard Exceptions in python

- ***Exception***: Base class for all exceptions
- ***StopIteration***: raised when the next() method of an iterator does not point to any object.
- ***SystemExit***: Exception that occurs when we want to stop the program execution and raise an error. If not handled in the code, interpreter exits.
- ***StandardError***: Base class for all built-in exceptions except StopIteration and SystemExit.

CONT...

- ***ArithmeticError***: Base class for all errors that occur for numeric calculation.
- ***OverflowError***: Raised when a calculation exceeds maximum limit for a numeric type.
- ***FloatingPointError***: Raised when a floating point calculation fails.
- ***ZeroDivisionError***: Raised when division or modulo by zero takes place for all numeric types.

CONT...

- ***AssertionError***: Raised in case of failure of the Assert statement. In assertion statement the user declares a condition to be true using assert statement prior to running the module.
- ***AttributeError***: Raised in case of failure of attribute reference or assignment.
- ***EOFError***: Raised when there is no input from the input() function and the end of file is reached.

CONT...

- ***ImportError***: Raised when an import statement fails.
- ***KeyboardInterrupt***: Raised when the user interrupts program execution.
- ***LookupError***: Base class for all lookup errors (IndexError, KeyError)i.e., raised when an index or a key is not found for a sequence or a dictionary respectively.
- ***IndexError***: Raised when an index is not found in a sequence.

CONT...

- ***KeyError***: Raised when the specified key is not found in the dictionary.
- ***NameError***: Raised when an identifier is not found in the local or global namespace.
- ***UnboundLocalError***: Raised when trying to access a local variable in a function or method but no value has been assigned to it.
- ***EnvironmentError***: Base class for errors that come from outside of Python. Parent class for IOError, OSError etc.

CONT...

- ***IOError***: Raised when an input/ output operation fails.
- ***OSError***: Raised for operating system-related errors.
- ***SyntaxError***: Raised when there is an error in Python syntax.
- ***IndendationError***: Raised when indentation is not specified properly.
- ***TypeError***: Raised when an operation or function is attempted that is invalid for the specified data type.

CONT...

- ***ValueError***: Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
- ***RuntimeError***: Raised when a generated error does not fall into any category.

FILE HANDLING IN PYTHON

In python, a file is categorized as either text or binary,and the difference between the two file open types is important.

Text files are structured as a sequence of lines,where each line includes a sequence of characters.

Python has several functions for creating, reading, updating, and deleting files.

open()

The key function for working with files in Python is the open()

function.

CNT...

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

CNT...

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

CNT...

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

read()

The read() method returns the whole text, but you can also specify how many characters you want to return.

CNT...

For example, to return the 5 first characters of the file :

```
f = open("demofile.txt", "r")

print(f.read(5))
```

You can return one line by using the readline() method:

```
f = open("demofile.txt", "r")

print(f.readline())
```

CNT...

By calling readline() two times, you can read the two first lines.

By looping through the lines of the file, you can read the whole file, line by line:

```
f = open("demofile.txt", "r")

for x in f:

  print(x)
```

CNT...

close()

Close the file when you are finish with it:

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

write

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

CNT...

Open the file "demo.txt" and append content to the file(demo file contains a sentence “hi,how are you” in it already):

```
f = open("demo.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demo.txt", "r")
print(f.read())
```

CNT...

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demo.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open("demo.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

CNT...

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

CNT...

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

A new empty file is created!

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

TECHOLAS
TECHNOLOGY DEMYSTIFIED

DIRECTORY MANIPULATION IN PYTHON

A directory or folder is a collection of files and sub-directories.

Python has the os module that provides useful methods to work with directories.

Following are the methods in the os module:

getcwd() - Get the present working directory

chdir() - Change the current working directory

CONT...

listdir() - List all the files and sub-directories inside a directory

mkdir() - make a new directory

rename() - Renaming a directory or file

remove() - Delete a file

rmdir() - Deletes an empty directory. It can’t delete a non-empty directory

CONT...

To remove a non-empty directory, rmtree() method from the shutil module need to be used.

```
import shutil

shutil.rmtree('directory_name')
```

CNT...

To Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

To delete an entire folder, use the os.rmdir() method:

```
import os

os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

TECHOLAS
TECHNOLOGY DEMYSTIFIED