Functions in Python

A function is set of statements that take input in the form of parameters, performs computation on the input and returns a result in the form of return statement

Syntax –

def `function-name` (`parameters if any`):

    # *function code*

    `return` statement

Note : It is a best practice to avoid usage of input() and print() functions in a function definition

WAF to calculate factorial of a number

In [15]:
```
# Function definition
def factorial(num) :
    fact = 1
    for i in range(num, 1, -1) :
        fact *= i
    return fact
```

In [16]:
```
# Function call
factorial("abcd")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 2
      1 # Function call
----> 2 factorial("abcd")

Cell In[15], line 4, in factorial(num)
      2 def factorial(num) :
      3     fact = 1
----> 4     for i in range(num, 1, -1) :
      5         fact *= i
      6     return fact

TypeError: 'str' object cannot be interpreted as an integer
```

In [22]:
```
# Function definition
def factorial(num) :
    if type(num) == int and num > 0 :
        fact = 1
        for i in range(num, 1, -1) :
            fact *= i
        return fact
    return "Invalid"
```

In [23]:
```
print(factorial("abcd"))
```

```
Invalid
```

Ex. Function returning multiple values as a tuple object

In [26]:
```
def func(num):
    sq = num**2
    cube = num**3
    return sq, cube   # packing of tuple - returning a tuple object
```

In [28]:
```
var1, var2 = func(5)  # unpacking the tuple
var1
```

Out[28]:
```
25
```

In [30]:
```
var = func(5)
var
```

Out[30]:
```
(25, 125)
```

Notes -

1. Function does not have any return type.
2. Parameters do not have data type. Developer has to validate the input to avoid errors.
3. A function returns None if return statement is not mentioned.
4. A function can have only 1 return statement and returns a single object

Function Arguments

- Required Positional Arguments
- Default Arguments
- Variable length Arguments
- Key-word Arguments
- Variable-length Keyword Arguments

Required Positional Argument

In [34]:
```
def demo(name, age) :
    print(f"Name - {name} | Age - {age}")

demo("Jane", 30)
demo(30, "Jane")
demo("Jane")
```

```
Name - Jane | Age - 30
Name - 30 | Age - Jane
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 6
      4 demo("Jane", 30)
      5 demo(30, "Jane")
----> 6 demo("Jane")

TypeError: demo() missing 1 required positional argument: 'age'
```

Examples -

In [41]:
```
help(list.insert)
```

```
Help on method_descriptor:

insert(self, index, object, /) unbound builtins.list method
    Insert object before index.
```

In [35]:
```
lst = [10, 20, 30, 40, 50, 60, 70, 80]
# Insert 5 at index 2
lst.insert(5, 2)
lst
```

Out[35]:
```
[10, 20, 30, 40, 50, 2, 60, 70, 80]
```

In [38]:
```
lst = [10, 20, 30, 40, 50, 60, 70, 80]
# Insert "abcd" at index 2
lst.insert("abcd", 2)
lst
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 3
      1 lst = [10, 20, 30, 40, 50, 60, 70, 80]
      2 # Insert "abcd" at index 2
----> 3 lst.insert("abcd", 2)
      4 lst

TypeError: 'str' object cannot be interpreted as an integer
```

Default Argument

In [44]:
```
def demo(name, age = 25) :
    print(f"Name - {name} | Age - {age}")

demo("Jane", 30)
demo(30, "Jane")
demo("Jane")
```

```
Name - Jane | Age - 30
Name - 30 | Age - Jane
Name - Jane | Age - 25
```

Examples -

In [40]:
```
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /) unbound builtins.str method
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

In [42]:
```
strg = "Mississippi"
strg.replace("i", "*")
```

Out[42]:
```
'M*ss*ss*pp*'
```

In [43]:
```
strg = "Mississippi"
strg.replace("i", "*", 2)
```

Out[43]:
```
'M*ss*ssippi'
```

In [47]:
```
help(sorted)
```

```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [45]:
```
lst = [1, 4, 3, 2]
sorted(lst)
```

Out[45]:
```
[1, 2, 3, 4]
```

In [46]:
```
lst = [1, 4, 3, 2]
sorted(lst, reverse=True)
```

Out[46]:
```
[4, 3, 2, 1]
```

Variable-length Argument

In [50]:
```
def demo(name, *args, age = 18) :
    print(f"Name - {name} | Age - {age} | Marks - {args}")

demo("Jane",80, 90, 70, 60, 50, 19)
```

```
Name - Jane | Age - 18 | Marks - ()
```

In [61]:
```python
def demo(name, *args, age = 18) :
    '''What does the function do'''
    if args :
        marks = [i for i in args if type(i) == int]  # for validation
        print(f"Name - {name} | Age - {age} | Total Marks - {sum(marks)}")
    else:
        print(f"Name - {name} | Age - {age} | Absent for exam")

demo("Jane",)
demo("Jane", 50, 60, 70, 80)
demo("Jane",50, 60, 70, "abcd")
```

```
Name - Jane | Age - 18 | Absent for exam
Name - Jane | Age - 18 | Total Marks - 260
Name - Jane | Age - 18 | Total Marks - 180
```

Key-word Argument

In [56]:
```python
demo("Jane", 50, 60, 70, 80, age = 19)
```

```
Name - Jane | Age - 19 | Total Marks - 260
```

In [62]:
```python
help(demo)
```

```
Help on function demo in module __main__:

demo(name, *args, age=18)
    What does the function do
```

Variable-length Keyword Argument

In [85]:
```python
def demo(name, *args, age = 18, **kwargs) :
    print(f"Name - {name} | Age - {age} | Marks - {args} | kwargs - {kwargs} | Mob

demo("Jane", 80, 90, 70, 60, 50, age = 19, gender = "F")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[85], line 4
      1 def demo(name, *args, age = 18, **kwargs) :
      2     print(f"Name - {name} | Age - {age} | Marks - {args} | kwargs - {kwargs}
| Mob - {kwargs["mob"]}")
----> 4 demo("Jane", 80, 90, 70, 60, 50, age = 19, gender = "F")

Cell In[85], line 2, in demo(name, age, *args, **kwargs)
      1 def demo(name, *args, age = 18, **kwargs) :
----> 2     print(f"Name - {name} | Age - {age} | Marks - {args} | kwargs - {kwargs}
| Mob - {kwargs["mob"]}")

KeyError: 'mob'
```

In [77]:
```python
def demo(name, *args, **kwargs) :
    print(f"Name - {name} | Marks - {args} | kwargs - {kwargs}")
```

```
demo("Jane", 80, 90, 70, 60, 50, age = 19, gender = "F", mob = 98765432)
```

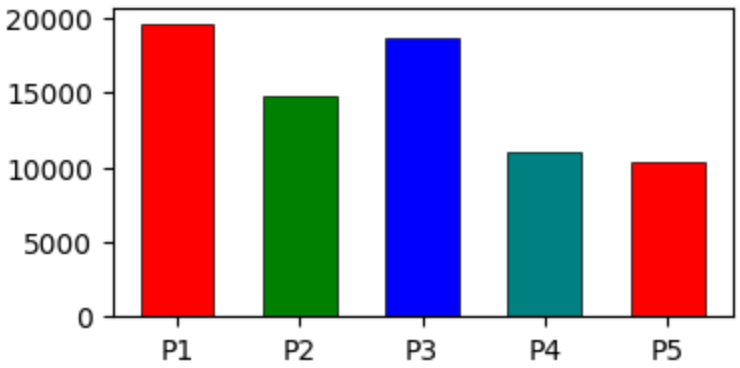
```
Name - Jane | Marks - (80, 90, 70, 60, 50) | kwargs - {'age': 19, 'gender': 'F', 'mo
b': 98765432}
```

Example -

In [87]:
```
import matplotlib.pyplot as plt
import random as r
plt.figure(figsize=(4,2))
products = ["P1", "P2", "P3", "P4", "P5"]
sales = [r.randint(10000, 20000) for i in range(5)]

plt.bar(color = ["red", "green", "blue", "teal"], width = 0.6, x = products, edgeco
```

Out[87]: `<BarContainer object of 5 artists>`



Significance of `/` and `*` in function definition

- `/` - All the arguments before `/` must be positional only
- `*` - All the arguments after `*` must be key-word only

if `*` without / then all arguments before * can be either positional or key-word

In [88]:
```
def demo(name, age = 25) :
      print(f"Name - {name} | Age - {age}")

demo("Jane", 30)  # positional only
demo(30, "Jane")  # positional only
demo(name = "Jane", age = 30)  # key-word only
demo(age = 30, name = "Jane")  # key-word only
```

```
Name - Jane | Age - 30
Name - 30 | Age - Jane
Name - Jane | Age - 30
Name - Jane | Age - 30
```

In [89]:
```
def demo(name, /, age = 25) :
      print(f"Name - {name} | Age - {age}")

demo("Jane", 30)
```

```
demo("Jane", age = 30)
demo(name = "Jane", age = 30)
```

```
Name - Jane | Age - 30
Name - Jane | Age - 30
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[89], line 6
      4 demo("Jane", 30)
      5 demo("Jane", age = 30)
----> 6 demo(name = "Jane", age = 30)

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame'
```

In [90]:
```
help(str.replace)
```

```
Help on method_descriptor:

replace(self, old, new, count=-1, /) unbound builtins.str method
    Return a copy with all occurrences of substring old replaced by new.

      count
        Maximum number of occurrences to replace.
        -1 (the default value) means replace all occurrences.

    If the optional argument count is given, only the first count occurrences are
    replaced.
```

In [92]:
```
strg = "Mississippi"
strg.replace("i", "*", 2)
strg.replace("i", "*", count = 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[92], line 3
      1 strg = "Mississippi"
      2 strg.replace("i", "*", 2)
----> 3 strg.replace("i", "*", count = 2)

TypeError: str.replace() takes no keyword arguments
```

In [94]:
```
def demo(name, /, *, age = 25) :
      print(f"Name - {name} | Age - {age}")

demo("Jane", age = 30)
demo(name = "Jane", age = 30)
demo("Jane", 30)
```

```
Name - Jane | Age - 30
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[94], line 5
      2       print(f"Name - {name} | Age - {age}")
      4 demo("Jane", age = 30)
----> 5 demo(name = "Jane", age = 30)
      6 demo("Jane", 30)

TypeError: demo() got some positional-only arguments passed as keyword arguments: 'n
ame'
```

In [95]:
```
help(sorted)
```

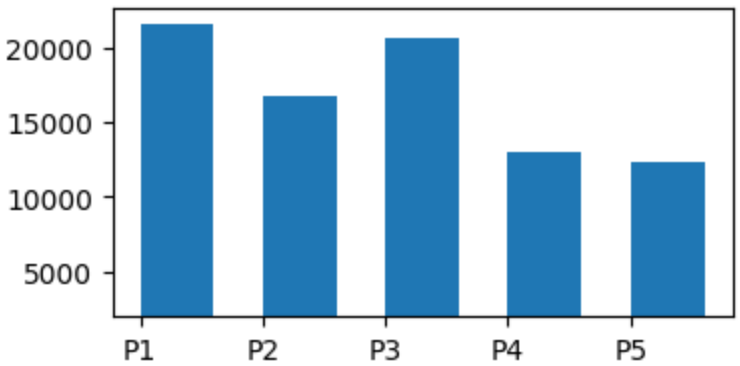
```
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

In [101…
```
plt.figure(figsize=(4, 2))
plt.bar(products, sales, 0.6, 2000, align="edge")
```

Out[101… `<BarContainer object of 5 artists>`



Example on packing and unpacking of tuples wrt to function call

In [49]:
```
def demo(name, *args, age = 18) :
    print(f"Name - {name} | Age - {age} | Marks - {args}")

demo("Jane", 80, 90, 70, 60, 50, 19)
```

```
Name - Jane | Age - 18 | Marks - (80, 90, 70, 60, 50, 19)
```

Ex. Using function write_to_file copy all the contents of a list/dict to the file

In [117…
```
def write_to_file(name, gender, age) :
    with open("details.txt", "a") as file :
        file.write(f" Name - {name} | Age - {age} | Gender - {gender}\n")
        print(f"Details for {name} added to the file")
```

Ex - using list of tuples

In [118…
```
data = [("Rosie", 30, "F"), ("Jack", 35, "M"), ("Anne", 25, "F")]
```

In [119…
```
for tup in data :
    write_to_file(*tup) # unpacking of tuples
```

```
Details for Rosie added to the file
Details for Jack added to the file
Details for Anne added to the file
```

Ex. using list of dict

In [120…
```
keys = ["name", "age", "gender"]
data = [dict(zip(keys, tup)) for tup in data]
data
```

Out[120…
```
[{'name': 'Rosie', 'age': 30, 'gender': 'F'},
 {'name': 'Jack', 'age': 35, 'gender': 'M'},
 {'name': 'Anne', 'age': 25, 'gender': 'F'}]
```

In [121…
```
for dct in data :
    write_to_file(**dct) # unpacking of dict
```

```
Details for Rosie added to the file
Details for Jack added to the file
Details for Anne added to the file
```

Variable Scope

- A namespace is a Container where names are mapped to variables
- A scope defines the hierarchical order in which the namespaces need to be
- searched in order to obtain the name-to-object mapping
- Scope defined the accessibility and lifetime of a variable

The LEGB rule

`The LEGB` rule decides the order in which the namespaces are to be searched for variable scoping

Variable scope hierarchy:

1. Built-In (B): Reserved names in Python
2. Global Variable (G): Defined at the uppermost level
3. Enclosed (E): Defined inside enclosing or nested functions
4. Local Variable (L): Defined inside a function

In [124…
```
def outer():
    # a = 7   # enclosed scope
    def inner() :
        # a = 10  # local variable
        print(a)
```

```
    inner()

# Main env
a = 5  # Global scope
outer()
```

```
5
```

In [5]:
```
from math import pi   # built-in scope
def outer():
    # pi = 7   # enclosed scope
    def inner() :
        pi = 10  # local variable
        z = 15
        print("pi in inner - ", pi)
    inner()
    print(z)

# Main env
outer()
inner()
print("pi in main - ", pi)
```

```
pi in inner -  10
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[5], line 12
      9     print(z)
     11 # Main env
---> 12 outer()
     13 inner()
     14 print("pi in main - ", pi)

Cell In[5], line 9, in outer()
      7     print("pi in inner - ", pi)
      8 inner()
----> 9 print(z)

NameError: name 'z' is not defined
```

In [135…

In [12]:
```
def func():
    global a, b  # marking variable a as global variable
    a = 10  # defining/modifying the variable
    b = 5
    print(a)  # using the variable

a = 15
print(a)
func()
print(a)
print(b)
```

```
15
10
10
5
```

Lambda Function

- A lambda function is also called as an anonymous function as it is a function that is defined without a name.
- A lambda function behaves similar to a standard function except it is defined in one-line.
- It is defined using a lambda key-word.
- 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.
- Syntax of Lambda Function –

  lambda *parameters* : *expression*

Write a lambda function to return addition of 2 numbers

In [13]:
```
lambda num1, num2 : num1 + num2
```

Out[13]:
```
<function __main__.<lambda>(num1, num2)>
```

In [15]:
```
add = lambda num1, num2 : num1 + num2
add(2, 5)
```

Out[15]:
```
7
```

Write a lambda function to return square of the number

In [16]:
```
square = lambda num : num ** 2
square(5)
```

Out[16]:
```
25
```

Function Object

- Everything in Python is an object, including functions.
- You can assign them to variables, store them in data structures, and pass or return them to and from other functions
- Functions in Python can be passed as arguments to other functions, assigned to variables or even stored as elements in various data structures.

function definition/implemenation

In [17]:
```
def func(a, b):  # -> function definition
    if a < b :
        return a
    else:
        return b
```

function call

In [18]:
```
# function call
var = func(2, 3)
var
```

Out[18]: 2

function object

In [19]:
```
# function object
var = func
var
```

Out[19]: <function __main__.func(a, b)>

In [20]:
```
var(2, 3)
```

Out[20]: 2

In [21]:
```
var = len
```

In [22]:
```
var("abcd")
```

Out[22]: 4

Applications of Function Object

Ex. WAP to sort a list of strings as per the last character.

In [23]:
```
lst = ["train", "car", "flight", "bike"]
sorted(lst)  # sorted alphabetically
```

Out[23]: ['bike', 'car', 'flight', 'train']

In [26]:
```
sorted(lst, key = len)  # sorted by len
```

Out[26]: ['car', 'bike', 'train', 'flight']

In [28]:
```
sorted(lst, key = lambda strg : strg[-1])  # sorted by last char
```

Out[28]: ['bike', 'train', 'car', 'flight']

In [30]:
```
max(lst, key = len)
```

Out[30]: 'flight'

In [32]:
```
min(lst, key = min)
```

Out[32]: 'train'

Note -

- There are many functions like sorted which take function object as an argument.
- Use a built-in function if available, else defined a custom function.
- If the logic for custom function is one-liner use lambda function else use standard user-defined function

In [ ]:
```
def sorted(iterables, key = None, reverse = False) :
    if key :
        temp = []
        for i in iterables :
            temp.append(key(i))
        # sort the temp and apply the sort order to original iterable

    else:
        # apply defualt sorting

    return list_obj
```

In [ ]:
```

```

In [ ]:
```
[expression for i in seq if condition]
```

In [ ]:
```
def func1():
    # some task for 4 lines
```

In [33]:
```
def func(*args):
    marks = [func1(i) for i in args if type(i) == int]  # filtering int elements
    print(sum(marks))
```

1. Comprehension is faster than map()-filter.
2. Comprehension performs both map() and filter tasks in a single statement
3. Use map()/filter() when expression consists of multiple lines

map() - filter() - reduce()

map(func_obj, sequence)

- func_obj : It is a function object to which map passes each element of given sequence.
- sequence : It is a sequence which is to be mapped.

- Returns a sequence of the results after applying the given function to each item of a given iterable.

Ex. WAP to map the given list to list of squares

In [35]:
```
lst = [1, 2, 3, 4]
list(map(lambda num : num**2, lst))
```

Out[35]:
```
[1, 4, 9, 16]
```

Ex. WAP to generate a list of squares of the given list using map

In [36]:
```
import math
squares = [1, 4, 9, 16]
list(map(math.sqrt, squares))
```

Out[36]:
```
[1.0, 2.0, 3.0, 4.0]
```

filter(func_obj, sequence)

- func_obj : function that tests if each element of a sequence true or not. It should always be a Boolean function.
- sequence : It is a sequence which is to be filtered.
- Returns a sequence of filtered elements.

Ex. WAP to filter all the even numbers from the given list of numbers.

In [38]:
```
lst = [1, 2, 4, 3, 5, 7, 6]
list(filter(lambda num : num % 2 == 0, lst))
```

Out[38]:
```
[2, 4, 6]
```

In [39]:
```
sum(filter(lambda num : num % 2 == 0, lst))
```

Out[39]:
```
12
```

In [40]:
```
len(filter(lambda num : num % 2 == 0, lst))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 len(filter(lambda num : num % 2 == 0, lst))

TypeError: object of type 'filter' has no len()
```

In [42]:
```
min(filter(lambda num : num % 2 == 0, lst))
```

Out[42]:
```
2
```

In [43]:
```
sorted(filter(lambda num : num % 2 == 0, lst))
```

Out[43]: [2, 4, 6]

In [41]:
```
len(tuple(filter(lambda num : num % 2 == 0, lst)))
```

Out[41]: 3

reduce(func_obj, seq)

- The reduce(func_obj, seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along. This function is defined in “functools” module.
  Note: func_obj will always take two parameters

Working :

- At first step, first two elements of sequence are picked and the result is obtained.
- Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.
- This process continues till no more elements are left in the container.
- The final returned result is returned and printed on console.

Ex. Reduce the list to summation of all numbers

In [45]:
```
from functools import reduce
reduce(lambda x, y : x + y, range(1, 6))
```

Out[45]: 15

In [46]:
```
from functools import reduce
reduce(lambda x, y : x + y, range(1, 6), 10)
```

Out[46]: 25

Additional Examples

Ex. Factorial using reduce

In [47]:
```
num = 5
reduce(lambda x, y : x * y, range(1, num+1))
```

Out[47]: 120

Ex. WAP to display the student details in sorted order of their marks.

In [51]:
```
students = {"Jane" : 40, "Max" : 50, "Sam" : 45, "Mary" : 70}
dict(sorted(students.items(), key = lambda val : val[1], reverse=True))
```

Out[51]: {'Mary': 70, 'Max': 50, 'Sam': 45, 'Jane': 40}

In [52]:
```
students.items()
```

Out[52]:
```
dict_items([('Jane', 40), ('Max', 50), ('Sam', 45), ('Mary', 70)])
```

Ex. WAP to extract all digits from a string (using filter)

In [53]:
```
string = "abcd1234"
list(filter(lambda ch : ch.isdigit(), string))
```

Out[53]:
```
['1', '2', '3', '4']
```

In [55]:
```
str.isdigit("123")
```

Out[55]:
```
True
```

In [56]:
```
string = "abcd1234"
list(filter(str.isdigit, string))
```

Out[56]:
```
['1', '2', '3', '4']
```

Ex. use reduce() to extract sum of all digits in a string

In [61]:
```
string = "abcd1234"
reduce(lambda x, y : x + (int(y) if y.isdigit() else 0), string, 0)
```

Out[61]:
```
10
```

In [62]:
```
string="shdghsg123djhjdsh76"
reduce(lambda x,y : x + (int(y) if y.isdigit() else 0), string, 0)
```

Out[62]:
```
19
```

Ex. Using map() multiply the two lists

In [64]:
```
lst1 = [1, 2, 3, 4, 5]
lst2 = [10, 20, 30, 40]
list(map(lambda x, y : x * y, lst1, lst2))
```

Out[64]:
```
[10, 40, 90, 160]
```

WAP to print sum of squares of numbers in range(1, 6)

In [66]:
```
reduce(lambda x, y : x + y**2, range(1, 6), 0)
```

Out[66]:
```
55
```

Fibonacci Series -

0 1 1 3 5 8 13 21 34 ....

In [67]:
```
fibo = [0, 1]
for i in range(8) :
```

```
    fibo.append(fibo[-1] + fibo[-2])
fibo
```

Out[67]: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

In [70]:
```
reduce(lambda x, y : x + [x[-1] + x[-2]],range(8), [0, 1])

[0, 1, 1] + [1+1]
```

Out[70]: [0, 1, 1, 2]

Decorators

In [73]:
```
def outer(a):
    def inner(b):
        return a + b
    return inner(5)

var = outer(10)
var
```

Out[73]: 15

In [75]:
```
def outer(a):
    def inner(b):
        return a + b
    return inner

var = outer(10)
var
```

Out[75]: <function __main__.outer.<locals>.inner(b)>

In [76]:
```
var(10)
```

Out[76]: 20

In [77]:
```
var(5)
```

Out[77]: 15

In [86]:
```
def get_file(file_name = None) :
    with open(file_name) as file :
        data = file.readlines()
        print("Reading file")

    print("Data clening formed as file is read")

    def get_filtered_data1(condition) :
        #filter(lambda : _________, data)
        return f"Returing filtered data for {condition}"

    def get_filtered_data2(condition) :
```

```
        #filter(lambda : _________, data)
        return f"Returing filtered data for {condition}"

    def get_filtered_data3(condition) :
        #filter(lambda : _________, data)
        return f"Returing filtered data for {condition}"

    return get_filtered_data1, get_filtered_data2, get_filtered_data3
```

In [90]:
```
f1, f2, f3 = get_file("customers.txt")
f1
```

```
Reading file
Data clening formed as file is read
```

Out[90]:
```
<function __main__.get_file.<locals>.get_filtered_data1(condition)>
```

In [88]:
```
resultset[0]("Managers")
```

Out[88]:
```
'Returing filtered data for Managers'
```

In [84]:
```
resultset("Developers")
```

Out[84]:
```
'Returing filtered data for Developers'
```

In [107…
```
# Decorator
def is_int(func) :
    def check(*params) :
        if all(map(lambda x : type(x) == int, params)) :
            return func(*params)
        else:
            return "Invalid"
    return check

@is_int
def factorial(num) :
    fact = 1
    for i in range(num, 1, -1):
        fact *= i
    return fact

@is_int
def even_odd(num) :
    return "even" if num % 2 == 0 else "odd"

@is_int
def add(num1, num2) :
    return num1 + num2

add(5, 3)
```

Out[107…
```
8
```

In [108…
```
factorial(5)
```

Out[108… 120

Exception Handling

It may be convenient to recognize the problems in your python code before you put it to real use. But that does not always happen. Sometimes, problems show up when you run the code; sometimes, midway of that. A Python exception is an error that's detected during execution.

Python does not provide any compile time Exception Handling. Developer has to proactively recognize the need for exception handling.

- What are Errors and Exceptions
- Handling Exceptions
- Defining Clean-up Actions
- Predefined Clean-up Actions
- Raising Exceptions

Syntax Error

In [109…
```
for i in range(5)
   print(i)
```

```
  Cell In[109], line 1
    for i in range(5)
                      ^
SyntaxError: expected ':'
```

Exception

Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it. Errors detected during execution are called exceptions and are not unconditionally fatal

ZeroDivisionError

In [110…
```
a, b = 1, 0
print(a/b)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[110], line 2
      1 a, b = 1, 0
----> 2 print(a/b)

ZeroDivisionError: division by zero
```

ValueError

In [111…
```
int("abcd")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[111], line 1
----> 1 int("abcd")

ValueError: invalid literal for int() with base 10: 'abcd'
```

NameError

In [112…
```
print(z)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[112], line 1
----> 1 print(z)

NameError: name 'z' is not defined
```

FileNotFoundError

In [113…
```
open("abc.txt")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[113], line 1
----> 1 open("abc.txt")

File ~\AppData\Local\Programs\Python\Python312\Lib\site-packages\IPython\core\intera
ctiveshell.py:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what you are doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'abc.txt'
```

TypeError

In [114…
```
"2" + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[114], line 1
----> 1 "2" + 2

TypeError: can only concatenate str (not "int") to str
```

In [115…
```
def func(a):
    pass

func()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[115], line 4
      1 def func(a):
      2     pass
----> 4 func()

TypeError: func() missing 1 required positional argument: 'a'
```

IndexError

In [116…
```
l = [1,2,3]
l[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[116], line 2
      1 l = [1,2,3]
----> 2 l[10]

IndexError: list index out of range
```

KeyError

In [117…
```
d = {1:2, 3:4}

d["abc"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[117], line 3
      1 d = {1:2, 3:4}
----> 3 d["abc"]

KeyError: 'abc'
```

ModuleNotFoundError

In [118…
```
import math1
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[118], line 1
----> 1 import math1

ModuleNotFoundError: No module named 'math1'
```

AttributeError

In [119…
```
import math
math.sq1()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[119], line 2
      1 import math
----> 2 math.sq1()

AttributeError: module 'math' has no attribute 'sq1'
```

In [120…
```
string = "abcd"

string.UPPER()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[120], line 3
      1 string = "abcd"
----> 3 string.UPPER()

AttributeError: 'str' object has no attribute 'UPPER'
```

Handling Exceptions

try :

#risky code

except :

```
    #code to handle error
```

else:

#executed if everything goes fine

finally:

#gets executed in either case.

In [123…
```
try :
    file = open("customers1.txt")
    data = file.read()
```

```
    print("Try block success")
except Exception as e :
    print(e)
else :
    file.close()
    print("Else block")
finally :
    print("Finally")
```

```
[Errno 2] No such file or directory: 'customers1.txt'
Finally
```

Rasie an Exception

Ex. WAP to create a function which takes and validates int input in the range of 0 - 9

In [127…

```
def get_input() :
    try :
        num = int(input("Enter a number in range of 0 - 9 : "))
        if 0 <= num <= 9 :
            return num
        else :
            raise TypeError("number must be in range of 0 - 9")

    except ValueError as  e :
        print("Value must be int")
        return get_input()

    except Exception as e :
        print(e)
        return get_input()
```

In [128…

```
get_input()
```

```
number must be in range of 0 - 9
Value must be int
```

Out[128… 2

Object Oriented Programming

- `Object-oriented programming` is a programming methdology that provides a means of structuring programs so that properties and behaviors are encapsulated into `individual objects` .
- For instance, an object could represent a person with properties like a name, age, and address and behaviors such as walking, talking, breathing, and running. Or it could

represent an email with properties like a recipient list, subject, and body and behaviors like adding attachments and sending.

Define a Class in Python

- A class definition starts with the `class` keyword, which is followed by the name of the class and a colon.
- Any code that is indented below the class definition is considered part of the class's body.

- The attributes that objects must have are defined in a `__init__()`. It is called as `Constructor` of the class.
- Every time a new object is created, `__init__()` sets the initial state of the object by assigning the values of the object's properties.
- `__init__()` initializes each new instance of the class.
- Attributes created in `__init__()` are called **instance attributes**. An instance attribute's value is specific to a particular instance of the class.
- Instance attributes are always referred using `self`.
- `Instance methods` are functions that are defined inside a class and can only be called from an instance of that class.

- A `class attribute` is always defined outside the constructor and always referred using class name.

Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

Examples -

Ex. Create class Circle

In [136…

```
class Circle :
```

```
    def __init__(self) :
        self.radius = 0

    def cal_area(self) :
        self.area = 3.14 * (self.radius ** 2)

c = Circle()
print(c)
print(c.radius)
c.radius = 10
# print(c.area)  # Error
c.cal_area()
print(c.area)
```

```
<__main__.Circle object at 0x00000155C7B1BD70>
0
314.0
```

In [156…
```
class Circle :

    def __init__(obj, radius) :
        obj.radius = radius

    def cal_area(obj) :
        obj.area = 3.14 * (obj.radius ** 2)
        return obj.area

c = Circle(10)
c.cal_area()
print(c.area)
```

```
314.0
```

In [158…
```
Circle.cal_area(c)
```

Out[158…
```
314.0
```

In [159…
```
circles = [Circle(10), Circle(20), Circle(100)]
[Circle.cal_area(c) for c in circles]  # Comprehesnsion
```

Out[159…
```
[314.0, 1256.0, 31400.0]
```

In [160…
```
list(map(Circle.cal_area, circles))
```

Out[160…
```
[314.0, 1256.0, 31400.0]
```

In [ ]:
```

```

In [151…
```
"123".isdigit()
```

Out[151…
```
True
```

In [152…
```
str.isdigit("123")
```

Out[152…
```
True
```

In [141…
```
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'a
sin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'fact
orial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'lo
g1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'r
emainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

In [142…
```
print(dir(c))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduc
e__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subcla
sshook__', '__weakref__', 'area', 'cal_area']
```

Ex. Create parent class `Shape` and `Circle`, `Rectangle`, `Triangle` as its child classes.

In [ ]:
```

```

Ex. Define `cal_area()` as abstract method in Shape class.

In [ ]:
```

```

Ex. Override `cal_area()` in all child classes of Shape class.

In [ ]:
```

```

Ex. Define `color_cost()` method in Shape class. Define **kwargs in constructor of all child classes to set shape color at the time of object creation.

In [ ]:
```

```

Multiple Inheritance and Method Resolution Order

In [ ]:
```

```
