Functions

```
* Built Functions like print, len, sum, str, int, float, type, ran
ge
* Function Declaration syntax:
    def function_name():
        statement
        return 0
```

In [2]:
```
# Delcaration of function

def add(x,y): # parameters
    addition = x + y
    return addition
```

In [3]:
```
# CALL function

add(5,4) # postional arguments
```

Out[3]: 9

In [4]:
```
def sqr(x):
    return x**2, x*x
```

In [5]:
```
sqr(4)
```

Out[5]: (16, 16)

In [6]:
```
def test(a,b,e):
    return a - e, a + b
```

In [7]:
```
test(5,6,1)
```

Out[7]: (4, 11)

In [8]:
```
test(b = 6, a = 5, e = 1) # keyword arguments
```

Out[8]: (4, 11)

In [9]:
```
test(6, e = 1, b = 6 ) # mix arguments (first position argument then keywor
```

Out[9]: (5, 12)

In [10]:
```
def tax(salary, tax_rate):
    return salary* tax_rate
```

In [11]:
```
tax(50000, 0.10)
```

Out[11]: 5000.0

In [12]:
```
def tax(salary, tax_rate = 0.05): # Default arguments
    return salary* tax_rate
```

In [13]:
```
tax(10000)
```

Out[13]: 500.0

In [14]:
```
tax(10000, 0.10)
```

Out[14]: 1000.0

Lists

In [15]:
```
employee_name = "Ayesha"
employee_age = 24
employee_salary = 15000.0
```

In [16]:
```
employee1=[employee_name, employee_age, employee_salary]
```

In [18]:
```
employee1
```

Out[18]: ['Ayesha', 24, 15000.0]

In [19]:
```
type(employee1)
```

Out[19]: list

In [66]:
```
employee2 = ["Ali", 22, 45000.0]
employee3 = ["Aliza", 33000.0]
employee4 = ["Hafsa", 43000.0]
```

In [21]:
```
employee1.append("Male")
```

In [22]:
```
employee1
```

Out[22]: ['Ayesha', 24, 15000.0, 'Male']

In [24]:
```
employee1[-1]="Female"
```

In [26]:
```
employee1
```

Out[26]: ['Ayesha', 24, 15000.0, 'Female']

In [5]:
```
employee2.append("Male")
employee3.append("Female")
employee4.append("Female")
```

In [7]:
```
employee4
```

Out[7]:
```
['Hafsa', 43000.0, 'Female']
```

In [18]:
```
employee4[:2]
```

Out[18]:
```
['Hafsa', 43000.0]
```

We have 2 type of indexing

1. positive (that starts from 0), direction left to right
2. negative (that starts from -1), direction right to left

In [39]:
```
employee4[2]
```

Out[39]:
```
'Female'
```

In [40]:
```
employee4[-1]
```

Out[40]:
```
'Female'
```

In [43]:
```
employee4[:2]
```

Out[43]:
```
['Hafsa', 43000.0]
```

In [45]:
```
employee4.append("khi")
```

In [47]:
```
employee4
```

Out[47]:
```
['Hafsa', 43000.0, 'Female', 'khi', 'khi']
```

In [48]:
```
del employee4[-1]
```

In [50]:
```
employee4
```

Out[50]:
```
['Hafsa', 43000.0, 'Female', 'khi']
```

In [51]:
```
employee4.pop(-1)
```

Out[51]:
```
'khi'
```

In [53]:
```
employee4
```

Out[53]:
```
['Hafsa', 43000.0, 'Female']
```

In [22]:
```
employee4.insert(2,"BSE")
```

In [23]:
```
employee4
```

Out[23]:
```
['Hafsa', 43000.0, 'BSE', 'Female']
```

In [24]:
```
len(employee4)
```

Out[24]:
```
4
```

In [ ]:
```

```

List of list

In [26]:
```
lst = ["Danial",24,"Salesman",["Gulshan","Johar"]]
```

In [34]:
```
lst[-1][-1]
```

Out[34]:
```
'Johar'
```

In [32]:
```
len(lst[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
t)
Input In [32], in <cell line: 1>()
----> 1 len(lst[1])

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

In [58]:
```
lst[0]
```

Out[58]:
```
'Danial'
```

In [59]:
```
lst[-2]
```

Out[59]:
```
'Salesman'
```

In [61]:
```
lst[-1]
```

Out[61]:
```
['Gulshan', 'Johar']
```

In [63]:
```
lst[-1][0]
```

Out[63]:
```
'Gulshan'
```

In [36]:
```
employees = [['Ayesha', 24, 10000.0, 'Female'],employee2,employee3,employee
```

In [37]:
```
employees
```

Out[37]:
```
[['Ayesha', 24, 10000.0, 'Female'],
 ['Ali', 22, 45000.0, 'Male'],
 ['Aliza', 33000.0, 'Female'],
 ['Hafsa', 43000.0, 'BSE', 'Female']]
```

In [67]:
```
(employees[0][-2] + employees[1][2] + employees[2][1] + employees[-1][1])/4
```

Out[67]:
```
32750.0
```

In [50]:
```
employees
```

Out[50]:
```
[['Ayesha', 24, 10000.0, 'Female'],
 ['Ali', 22, 45000.0, 'Male'],
 ['Aliza', 33000.0, 'Female'],
 ['Hafsa', 43000.0, 'Female']]
```

In [51]:
```
age=[]
for item in employees[:2]:
    age.append(item[1])
```

In [53]:
```
age
```

Out[53]:
```
[24, 22]
```

In [57]:
```
for item in employees[2:]:
    item.insert(1,avg_age)
```

In [58]:
```
employees
```

Out[58]:
```
[['Ayesha', 24, 10000.0, 'Female'],
 ['Ali', 22, 45000.0, 'Male'],
 ['Aliza', 23.0, 33000.0, 'Female'],
 ['Hafsa', 23.0, 43000.0, 'Female']]
```

In [59]:
```
employees[-1][0]="marium"
```

In [60]:
```
employees
```

Out[60]:
```
[['Ayesha', 24, 10000.0, 'Female'],
 ['Ali', 22, 45000.0, 'Male'],
 ['Aliza', 23.0, 33000.0, 'Female'],
 ['marium', 23.0, 43000.0, 'Female']]
```

In [61]:
```
count_gender={}
for item in employees:
    gender=item[-1]
    if gender not in count_gender:
        count_gender[gender]=1
    else:
        count_gender[gender]+=1
```

In [62]:
```
count_gender
```

Out[62]:
```
{'Female': 3, 'Male': 1}
```

In [41]:
```
"e" in "ayesha"
```

Out[41]:
```
True
```

In [55]:
```
avg_age=sum(age)/len(age)
```

In [ ]:

In [ ]:

In [45]:
```
salaries = [] # [50000.0,45000.0]

for lst in employees:
    #print(lst[0], lst[-1])
    salary = lst[-2] # salary = 45000.0
    salaries.append(salary)
```

In [48]:
```
type(salaries[0])
```

Out[48]:
```
float
```

In [71]:
```
salaries
```

Out[71]:
```
[10000.0, 45000.0, 33000.0, 'BSE']
```

In [49]:
```
sum(salaries)/len(salaries)
```

Out[49]:
```
32750.0
```

In [38]:
```
employees[-1].pop(2)
```

Out[38]:
```
'BSE'
```

In [77]:
```
salaries = [] # [50000.0,45000.0]

for lst in employees:
    salary = lst[-2] # salary = 45000.0
    salaries.append(salary)
```

In [78]:
```
sum(salaries)/len(salaries)
```

Out[78]: 32750.0

Dictionary

In [79]:
```
employee_name = 'Bilal'
employee_age = 21
employee_salary = 20000.0
```

In [83]:
```
employee = [employee_name, employee_age, employee_salary]
print(employee)
print(type(employee))
```

```
['Bilal', 21, 20000.0]
<class 'list'>
```

In [85]:
```
employee = {'employee_name' : employee_name , 'employee_age' : employee_age
```

In [86]:
```
type(employee)
```

Out[86]: dict

In [87]:
```
employee['employee_name']
```

Out[87]: 'Bilal'

In [88]:
```
employee.keys()
```

Out[88]: dict_keys(['employee_name', 'employee_age', 'employee_salary'])

In [90]:
```
employee.values()
```

Out[90]: dict_values(['Bilal', 21, 20000.0])

In [91]:
```
# Membership operator - in
'employee_salary' in employee
```

Out[91]: True

In [92]:
```
21 in employee.values()
```

Out[92]: True

In [93]:
```
employee["employee_name"]="Bilal Khan"
employee
```

Out[93]:
```
{'employee_name': 'Bilal Khan', 'employee_age': 21, 'employee_salary': 200
00.0}
```

In [94]:
```
employee["employee_address"]="Karachi"
employee
```

Out[94]:
```
{'employee_name': 'Bilal Khan',
 'employee_age': 21,
 'employee_salary': 20000.0,
 'employee_address': 'Karachi'}
```

In [95]:
```
fruits = ['Apple',  'Mango','Apple', 'Oranges', 'Banana',
'Mango','Apple',  'Banana', 'Mango','Apple', 'Oranges',
'Apple','Oranges', 'Banana']
```

In [96]:
```
frq = {}

for i in fruits: # 'Apple'
    if i not in frq: # 'Apple' = False
        frq[i] = 1
    else:
        frq[i] += 1
```

In [97]:
```
frq
```

Out[97]:
```
{'Apple': 5, 'Mango': 3, 'Oranges': 3, 'Banana': 3}
```

Sets

- Sets are ordered collections of unique elements.
- Sets do not allow duplicate values.
- Sets are defined using curly braces { }

In [113]:
```
my_set = {"zara",9,"Ayesha", 2.2, 3.3, 2.2, 5}
print(my_set)
```

```
{2.2, 3.3, 5, 9, 'Ayesha', 'zara'}
```

In [112]:
```
type(my_set)
```

Out[112]:
```
set
```

In [107]:
```
my_set[0]="ALi"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [107], in <cell line: 1>()
----> 1 my_set[0]="ALi"

TypeError: 'set' object does not support item assignment
```

In [115]:
```
my_set.add("neww item")
```

In [116]:
```
my_set
```

Out[116]:
```
{2.2, 3.3, 5, 9, 'Ayesha', 'neww item', 'zara'}
```

In [118]:
```
my_set.remove('neww item')
```

In [120]:
```
my_set
```

Out[120]:
```
{2.2, 3.3, 5, 9, 'Ayesha', 'zara'}
```

Tuples

- Tuples are ordered, immutable collections of elements.
- Once created, you cannot change their contents.
- Tuples are defined using parentheses ( ), although the parentheses are often omitted.

In [104]:
```
my_tuple = (1,1, 'apple', 3.14)
print(my_tuple)
```

```
(1, 1, 'apple', 3.14)
```

In [105]:
```
my_tuple[0]=3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [105], in <cell line: 1>()
----> 1 my_tuple[0]=3

TypeError: 'tuple' object does not support item assignment
```

In [123]:
```
# Function to calculate percentage and grade
#write code for the task that will tae user input of names of three student
#along with marks of three courses . Calculate Marks obtained, Percentage a
#the grade and obtained marks in the list

def calculate_grade(total_marks):
    percentage = (total_marks / 300) * 100
    if percentage >= 90:
        return "A+", percentage
    elif 80 <= percentage < 90:
        return "A", percentage
    elif 70 <= percentage < 80:
        return "B", percentage
    elif 60 <= percentage < 70:
        return "C", percentage
    elif 50 <= percentage < 60:
        return "D", percentage
    else:
        return "F", percentage

# Input student names and marks
students = []
for i in range(3):
    student_name = input(f"Enter name of student {i + 1}: ")
    marks = []
    for j in range(3):
        course_marks = float(input(f"Enter marks for course {j + 1}: "))
        marks.append(course_marks)
    students.append({"name": student_name, "marks": marks})

# Calculate and display results for each student
for student in students:
    total_marks_obtained = sum(student["marks"])
    grade, percentage = calculate_grade(total_marks_obtained)

    print(f"\nStudent Name: {student['name']}")
    print(f"Total Marks Obtained: {total_marks_obtained}")
    print(f"Percentage: {percentage:.2f}%")
    print(f"Grade: {grade}")

print(students)
```

```
Enter name of student 1: ayesha
Enter marks for course 1: 45
Enter marks for course 2: 67
Enter marks for course 3: 87
Enter name of student 2: danial
Enter marks for course 1: 66
Enter marks for course 2: 78
Enter marks for course 3: 89
Enter name of student 3: reesha
Enter marks for course 1: 90
Enter marks for course 2: 55
Enter marks for course 3: 45

Student Name: ayesha
Total Marks Obtained: 199.0
Percentage: 66.33%
Grade: C

Student Name: danial
Total Marks Obtained: 233.0
Percentage: 77.67%
Grade: B

Student Name: reesha
Total Marks Obtained: 190.0
Percentage: 63.33%
Grade: C
[{'name': 'ayesha', 'marks': [45.0, 67.0, 87.0]}, {'name': 'danial', 'mark
s': [66.0, 78.0, 89.0]}, {'name': 'reesha', 'marks': [90.0, 55.0, 45.0]}]
```

In [124]:
```python
def find_second_largest(numbers):
    sorted_numbers = sorted(numbers, reverse=True)
    return sorted_numbers[1]

my_list = [3, 1, 7, 5, 9, 2]
second_largest = find_second_largest(my_list)
print(second_largest)  # Output: 7
```

```
7
```

List comprehension

In [63]:
```python
original_numbers = [1, 2, 3, 4, 5]

# Using a list comprehension to square each number
squared_numbers = [x ** 2 for x in original_numbers]

print(squared_numbers)  # Output: [1, 4, 9, 16, 25]
```

```
[1, 4, 9, 16, 25]
```

In [64]:
```
original_numbers = [1, 2, 3, 4, 5]
sq_lst=[]
for i in original_numbers:
    sq_lst.append(i*i)
```

In [65]:
```
sq_lst
```

Out[65]:
```
[1, 4, 9, 16, 25]
```

In [ ]:
```

```