upGrad

PLACEMENT REFRESHER PROGRAM

Session 2: Python II

Functions, OOP

By
Ritesh Kumar Pandey

Agenda

- Functions
  - Syntax
  - Arguments
  - Anonymous Functions
- OOP
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Function - Introduction

upGrad

What is a function and why to use it?

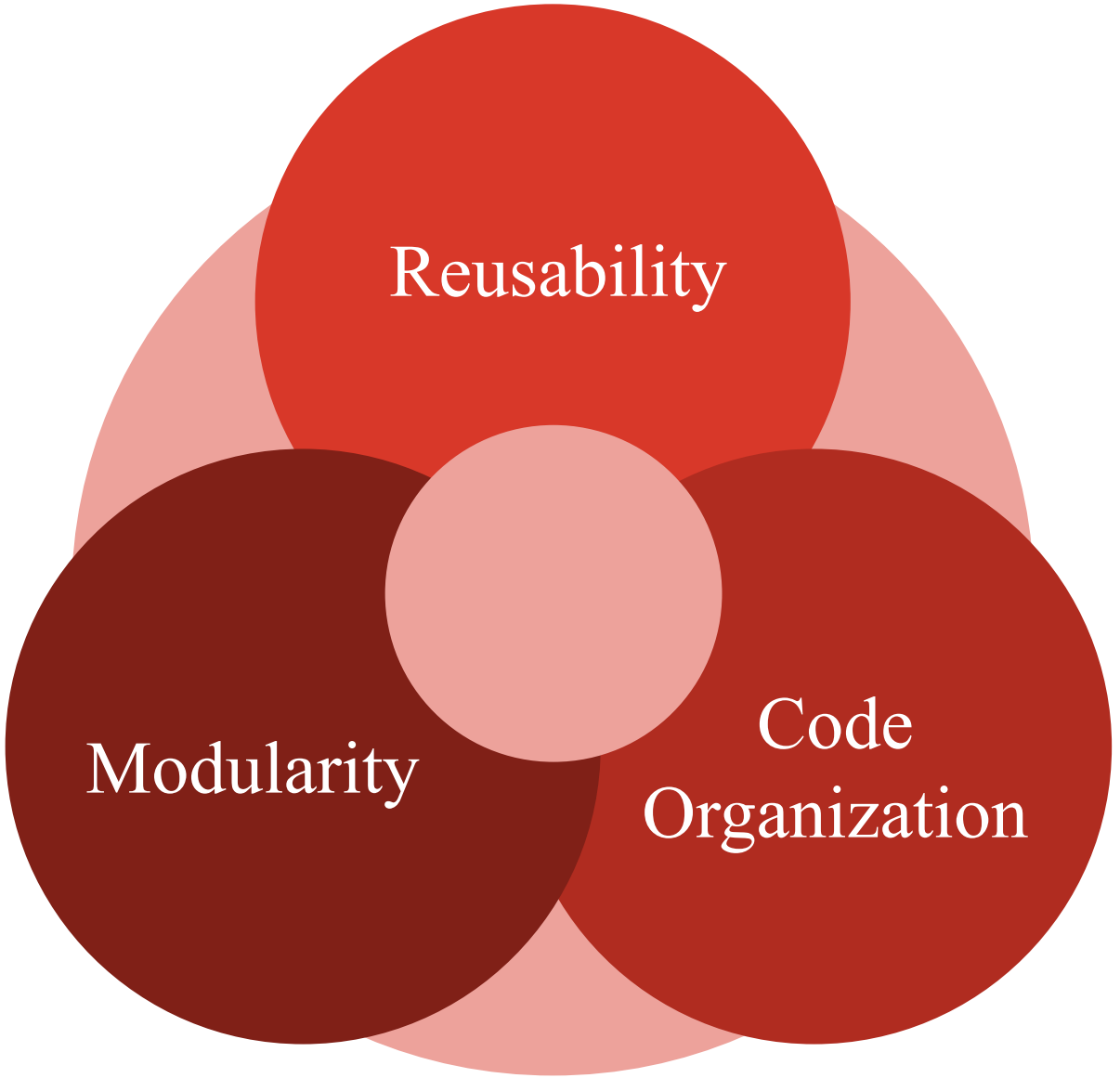
Function - Introduction

upGrad

What is a function and why to use it?

- building blocks that simplify code development and maintenance

- a block of reusable code that performs a specific task or set of tasks



Function - Types

upGrad

Which among the following is a valid function ?

1. def my_func():
   pass

2. abc()

3. print()

4. function xyz()

Function - Types

upGrad

Which among the following is a valid function ?

1. def my_func():
   pass

2. abc()

3. print()

4. function xyz()

Function - Types

upGrad

Which among the following is a valid function ?

1. def my_func():
   pass

User defined function

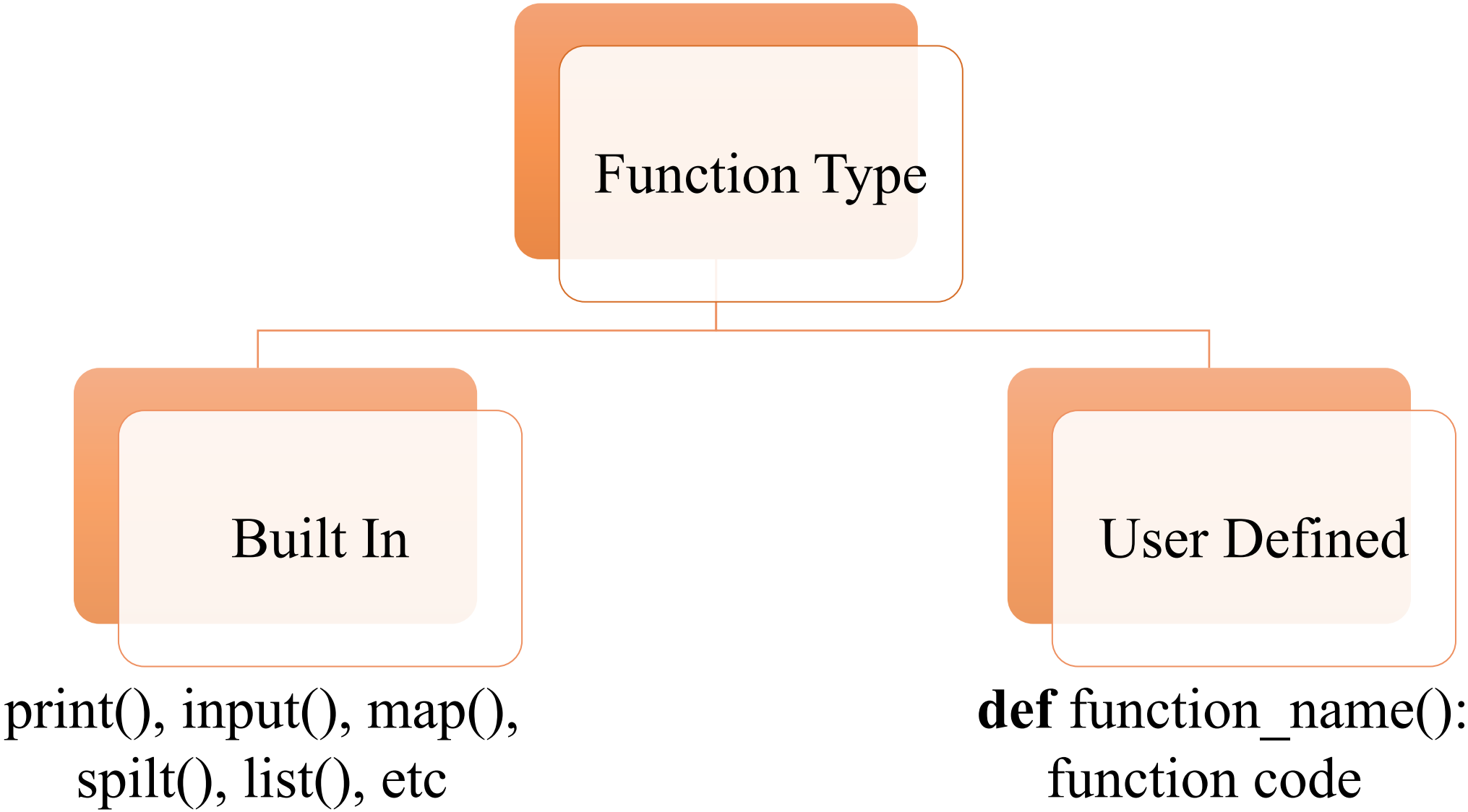
2. abc()

3. print()

In built function

4. function xyz()

Function - Types



Function - Syntax

upGrad

Which of the following is an valid function declaration ?

1. def my function_123():
   pass

2. def _myFunction123():
   pass

3. def 123myFunction():
   pass

4. def myFunction@123():
   pass

Function - Syntax

upGrad

Which of the following is an valid function declaration ?

1. def my function_123():
   pass

2. def _myFunction123():
   pass

3. def 123myFunction():
   pass

4. def myFunction@123():
   pass

Function - Syntax

upGrad

Which of the following is an valid function declaration ?

1. def my function_123():
   pass

   space not allowed

2. def _myFunction123():
   pass

3. def 123myFunction():
   pass

   can only start with a letter or underscore

4. def myFunction@123():
   pass

   can only contain letters, digits and underscore

Function - Syntax

upGrad

What is the output of the following code ?

```
x = 5.7
y = int(x)

def int(x):
    return x*10

z = int(x)
print(y)
print(z)
```

Function - Syntax

upGrad

What is the output of the following code ?

```
x = 5.7
y = int(x)

def int(x):
    return x*10

z = int(x)
print(y)
print(z)
```

```
5
57.0
```

Function - Arguments

upGrad

What is the output of the following code ?

```
def my_function(x, y=2):
    print('X =', x)
    print('Y =', y)
    print('X + Y =', x + y)

result_1 = my_function(5, 10)
result_2 = my_function(y=10, x=5)
result_3 = my_function(5)
```

Function - Arguments

upGrad

What is the output of the following code ?

```
def my_function(x, y=2):
    print('X =', x)
    print('Y =', y)
    print('X + Y =', x + y)

result_1 = my_function(5, 10)
result_2 = my_function(y=10, x=5)
result_3 = my_function(5)
```

```
X = 5
Y = 10
X + Y = 15
X = 5
Y = 10
X + Y = 15
X = 5
Y = 2
X + Y = 7
```

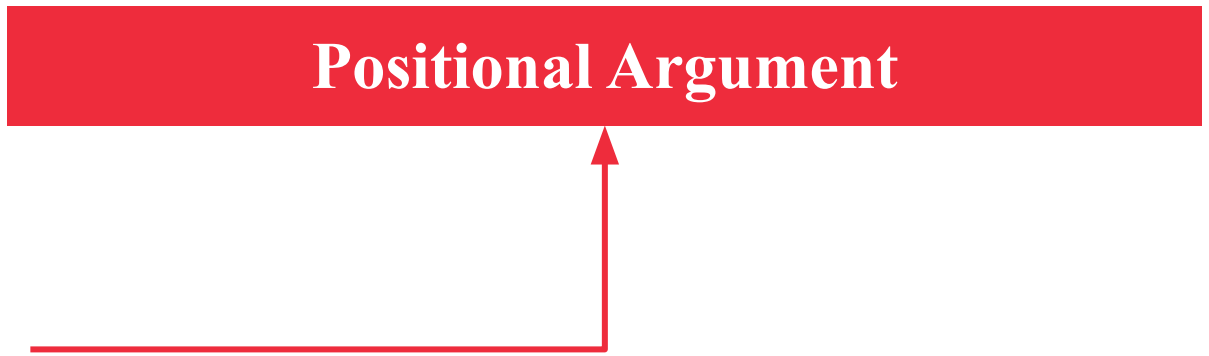
Function - Arguments

upGrad

What is the output of the following code ?

```
def my_function(x, y=2):
    print('X =', x)
    print('Y =', y)
    print('X + Y =', x + y)

result_1 = my_function(5, 10)
result_2 = my_function(y=10, x=5)
result_3 = my_function(5)
```



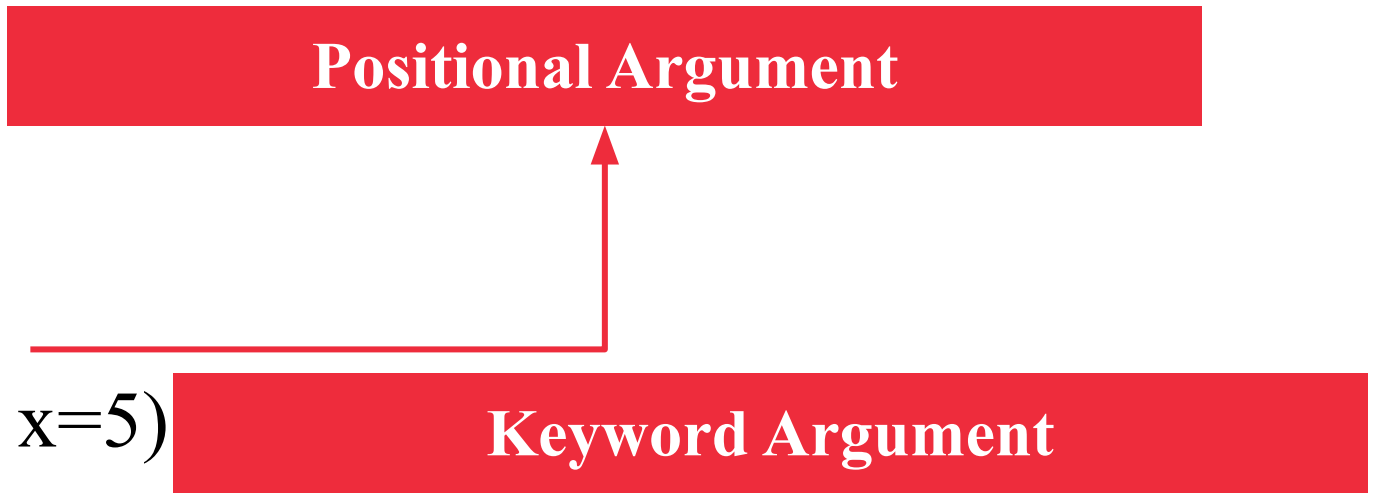
Function - Arguments

upGrad

What is the output of the following code ?

```
def my_function(x, y=2):
    print('X =', x)
    print('Y =', y)
    print('X + Y =', x + y)

result_1 = my_function(5, 10)
result_2 = my_function(y=10, x=5)
result_3 = my_function(5)
```



Function - Arguments

upGrad

What is the output of the following code ?

```
def my_function(x, y=2):
    print('X =', x)
    print('Y =', y)
    print('X + Y =', x + y)

result_1 = my_function(5, 10)
result_2 = my_function(y=10, x=5)
result_3 = my_function(5)
```

Positional Argument

Keyword Argument

Default Argument

Function - Arguments

upGrad

- The '*args` and '**kwargs' syntax in function definitions allows Python functions to accept a variable number of arguments

args (Arbitrary Positional Arguments):

- The *args syntax allows a function to accept an arbitrary number of positional arguments.
- In the function definition, *args is used as a parameter. A tuple is created by collecting any additional positional arguments provided during the function call.
- It is a convention to use args, but you can use any other name with the * prefix.

kwargs (Arbitrary Keyword Arguments):

- The **kwargs syntax allows a function to accept an arbitrary number of keyword arguments.
- In the function definition, **kwargs is used as a parameter. It collects any additional keyword arguments provided during the function call. The arguments are stored in a dictionary with their names as keys and their values as values.
- kwargs is a convention, but you can use any other name with the ** prefix.

Function - Arguments

upGrad

What is the output of the following code ?

```
def myfun(*args):
    print(type(args))

myfun(1,2,3)
myfun('Hii')
myfun([1, 2, 3])
```

Function - Arguments

upGrad

What is the output of the following code ?

```
def myfun(*args):
    print(type(args))

myfun(1,2,3)
myfun('Hii')
myfun([1, 2, 3])
```

```
<class 'tuple'>
<class 'tuple'>
<class 'tuple'>
```

Function - Scope of Variables

upGrad

What is the output of the following code ?

```
greeting = "Hello"
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
greeting = “Hello”
def message():
    global greeting
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

Function - Scope of Variables

upGrad

What is the output of the following code ?

```
greeting = "Hello"
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

Hii
Hello

```
greeting = “Hello”
def message():
    global greeting
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

Function - Scope of Variables

upGrad

What is the output of the following code ?

```
greeting = "Hello"
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
Hii
Hello
```

```
greeting = “Hello”
def message():
    global greeting
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
Hii
Hii
```

```
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

Function - Scope of Variables

upGrad

What is the output of the following code ?

```
greeting = "Hello"
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
Hii
Hello
```

```
greeting = “Hello”
def message():
    global greeting
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
Hii
Hii
```

```
def message():
    greeting = 'Hii'
    print(greeting)

message()
print(greeting)
```

```
Hi
ERROR !
```

Function - Scope of Variables

upGrad

Local scope:

- Variables declared within a function have local scope. They are accessible only within the function in which they are defined.
- Local variables are created when the function is called and destroyed when it exits.
- Local variables cannot be accessed from outside the function.

Global Scope:

- Variables declared outside of any function or at the top level of a script have global scope. They are accessible throughout the program.
- During program startup, global variables are created in memory and remain there until the program ends.
- Global variables can be accessed from any function within the program.

Function - Lambda Function

upGrad

Lambda Function

- Lambda functions, also known as anonymous functions, are small, unnamed functions defined in a single line.
- They are used when you need a simple function for a short, specific task.

Syntax:

- Lambda functions are defined using the lambda keyword, followed by parameters and an expression.
- Example: **lambda** x: x * 2 doubles the input x.

Function - Lambda Function

upGrad

Find the second-highest element in a list using lambda functions.

Input: [12, 45, 2, 41, 31, 10, 8, 6]

Output: 41

Function - Lambda Function

upGrad

Find the second-highest element in a list using lambda functions.

Input: [12, 45, 2, 41, 31, 10, 8, 6]

Output: 41

Hint: You can use the max() function with a custom key function (a lambda function) to find the second-highest element.

Function - Lambda Function

upGrad

Find the second-highest element in a list using lambda functions.

Input: [12, 45, 2, 41, 31, 10, 8, 6] **Output:** 41

```
def second_highest(numbers):
    if len(numbers) < 2:
        return None  # Not enough elements in the list

    # Use a lambda function to find the second-highest element
    second_max = max(numbers, key=lambda x: float('-inf') if x == max(numbers)
else x)

    return second_max
```

Object Oriented Programming

upGrad

- Object Oriented Programming (OOP) is a programming paradigm that uses **Objects** and **Classes** to structure and organize code.
- The primary goal of OOP is to **model real-world entities**, their **attributes**, and **interactions in software development**, leading to code that is more **modular, reusable,** and **easier to understand and maintain.**

- **Class:** A class is a blueprint or template for creating objects. It defines the structure (attributes/properties) and behavior (methods/functions) that the objects of that class will have.

- **Object:** An object is an instance of a class. It is a self-contained unit that can hold data and perform actions based on the class's blueprint.

Object Oriented Programming

upGrad

Abstraction

- hides unnecessary details while exposing only essential features

Encapsulation

- bundles the data (attributes) and methods (functions) that operate on that data within a single unit (object or class)

Inheritance

- allows one class to inherit the properties of another class

Polymorphism

- enables the use of the same method to behave differently based on the object being called

Suppose an **int** object occupies 4 bits of memory space, **str** object occupies 8 bits of memory space and a **list** object occupies 16 bits of memory space. Identify the memory space occupied by class student given in the following code.

```
class student:
    def __init__(self, name, rollno, marks, grade, subjects):
        self.name = name
        self.rollno = rollno
        self.marks = marks
        self.grade = grade
        self.subjects = subjects
```

name - str
rollno - int
marks - int
grade - str
subjects - list

Suppose an **int** object occupies 4 bits of memory space, **str** object occupies 8 bits of memory space and a **list** object occupies 16 bits of memory space. Identify the memory space occupied by class student given in the following code.

```
class student:
    def __init__(self, name, rollno, marks, grade, subjects):
        self.name = name
        self.rollno = rollno
        self.marks = marks
        self.grade = grade
        self.subjects = subjects
```

name - str
rollno - int
marks - int
grade - str
subjects - list

0

Class do not occupy memory space

THANK YOU