Functions:-

```
void sum (int a, int b)
{
    int sum1;
    sum1 = a+b;
    printf ("Sum = %d", sum1);
}
```

} func. defn.

sum(10,20) → func. call.

```
def sum(a,b):
    sum1 = a+b
    print(sum1)
```

} func. defn.

sum(10,20) ⟶ func. call.

Types of arguments:

1) Necessary
2) Default
3) Keyword
4) Variable length.

1) **Necessary:-**
Whatever arguments are specified in function definition, same order must be maintained even in the function call.

Eg:-
```
def display(name, age):
    print "Name =" + name
    print "Age =" + age
display ("Ravi", 20)
display (20, "Ravi")
```
⟶ this is wrong logically, since the order/type of argument does not match with fun. def.

display (20) ⟶ displays Error.

2) **Default:-**

Eg:-
```
def display(name, age = 20):
    print "Name =" + name
    print "Age" = " + age
display ("Ravi", 22)
display ("Ravi")
```
⟶ default value for age is considered.

→ Default arguments must always be specified at the end of the function definition.

3) **Keyword:-**

Eg:-
```
def display(age, name):
    print "Name =" + name
    print "Age =" + age
display(name = "Ravi", age = 20)
display(age = 20, name = "Ravi")
```
→ Order can be changed. Values are identified based on the keyword specified.

4) **Variable length:-**

Eg:-
```
def display(n1, *n2[]):
    print n1
    print n2
display(1, 2, 3, 4, 5)
```

... to define a function to sort the given elements using insertion sort technique.

→
```
n = input("Enter the number of elements :")
l = [ ]
for i in n:
    n1 = input("Enter the elements")
    l.append(n1)
```

Given 2 lists A and B of non-negative int ~~score~~ score. A Special ~~score~~ score is one which is a multiple of 10 such as 40 or 90. Return the sum of largest special score in A and largest special score in B. Write a PP or python function score_special to display the sum of largest special score in A and largest special score in B.

→
```
l1 = [15, 60, 80, 55, 65]
l2 = [22, 71, 90, 84, 32]
```

x = 10 → global

(1)

```
def incr():          → local
    x = 11
    x = x + 1
    print("After incr x =", x)   → 12
print("Before fun call x =", x)  → 10
incr()               → func call
print("After incr x =", x)       → 10
```

x = 10

```
def incr():          → keyword used to access global variables
    global x
    x = x + 1
    print("After incr x =", x)   → 11
print("Before incr x =", x)      → 10
incr()               → func call
print("After incr x =", x)       → 11
```

Local variable → if a variable is defined inside a function.

Global variable → if a variable is defined outside a function

Pass by value and Pass by reference:-

```
l = [1, 2, 3]
def display(a):
    b = [5, 6, 7]
```

```
a.append(b)
print a
display(l)
print l
```

```
x=10
def display(y):
    y=y+1
    print y
display(x)
print x
```

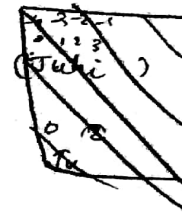
In python, there is no concept of pass by value.
In python, each and every item is treated as an object.
So when you pass an object, it means you are passing a reference.

If we pass mutable objects to functions, we can modify those objects. So the modifications or computations inside the function will affect the original memory location of the object.

→ A palindromic prime is a prime no. that is also a palindrome. For eg: 131 is a prime and also a palindromic prime. Write a program that displays the 1st hundred palindromic prime numbers.

→ A prime number is called Mersenne Prime if it can be written in the form $2^p - 1$ for some +ve integer p. Write a program that finds all Mersenne Prime with $p \leq 31$ and display the output as follows:

p	$2^p - 1$
2	3
3	7
5	31
⋮	⋮

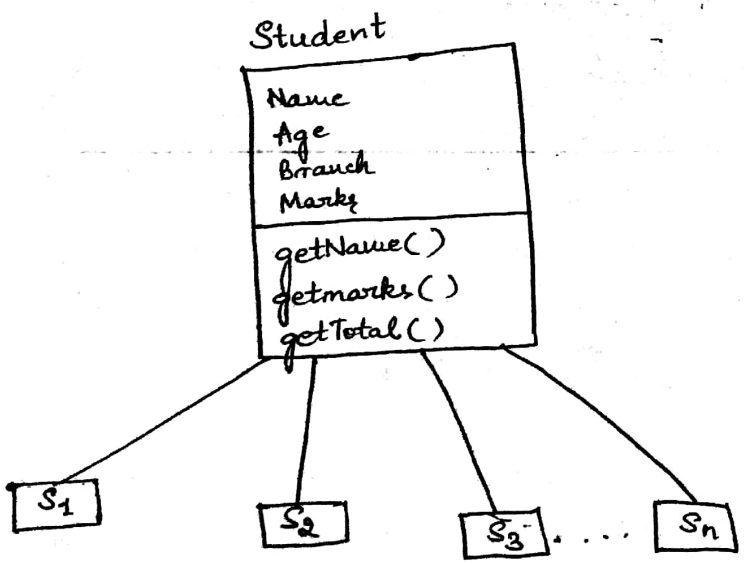


```
PP=[ ]
i=100
while (len(PP)<=100):
    if (str[
```

... pair of prime numbers that differ by 2 for eg. 3 and 5, 5 and 7, 11 and 13 are twin primes. Write a program to find all twin primes less than 1000.

o/p:- (3,5)
(5,7)
(11,13)
.
.
.

OOP:-



General Syntax:-

```
class class_name:
    // Attributes
    // Methods
```

Eg:- class Student:

```
    def __init__(self, name, age, branch):
        self.name = name
        self.branch = branch
        self.age = age

    def display(self):
        print("Name =", self.name)
        print("Age=", self.age)
        print("Branch =", self.branch)
```

Syntax:- object_name = class_name([arg])

Eg:-

```
s1 = Student()
s1 = Student('Ram', 20, 'CSE')
s1.display()
print(s1.name)
```

Dot operator is used to access the methods & attributes.

The functions or methods present inside the class must have argument 'self'.

In python, __init__ is a constructor.
Default values can also be specified for the attributes while defining the constructor (at the end of the definition).

Access specifiers :- public/private/protected.

In python, you can't specify the access specifiers.
By default, each and every attribute is public.
If you want to make it private, then use --

Eg:- --age ⇒ private attribute.

```
class rect:
    def __init__(self, h=10, w=20):
        self.h = h
        self.w = w
    def geth(self):
        print("H=", self.h)
    def getw(self):
        print("w=", self.w)
    def area(self):
        print("Area=", (self.h * self.w))

r1 = rect()
r2 = rect(20,40)
r1.area()
r2.area()
r1.getw()
r1.geth()
r2.geth()
r2.getw()
```

1) Design a class named account that contains,

(i) a private integer datafield named id for the account.

(ii) a private float datafield named balance for the account.

(iii) a private float datafield named annual interest rate that stores the current interest rate.

(iv) a constructor that creates an account with specified id (default=0), initial balance (default=100) & annual interest rate (default=0)

(v) The accessor & mutator methods for id, balance and annual interest rate.

(vi) A method named getMonthlyInterestRate() that returns the monthly interest rate.

(vii) A method named getMonthlyInterest() that returns the monthly interest.

(viii) A method named withdraw() that withdraws a specified amount from the account.

(ix) A method named deposit that deposits a specified amount in the account.

Monthly interest = Balance × Monthly interest rate.

Monthly interest rate = $\frac{\text{Annual interest rate}}{12}$

```
class Acc:
    __id = 20
    def getid(self):
        print("id =", self.__id)
    def setid(self, value):
        self.__id = value
    __balance = 100.0
    __annualInterestRate = 0.0
    def __init__(self, id=0, bal=100.0, amt=0.0)
        self.__id = id
        self.__balance = bal
        self.annualInterestRate = amt
    def getId(self)
        print(self.__id)
    def setId(self, id)
        self.__id = id
    def getBal(self)
        print(self.__balance)
    def setBal(self, bal)
        self.__balance = bal
```

hasattr(obj, attr) → will return true if the obj has the attribute specified.

getattr(obj, attr)

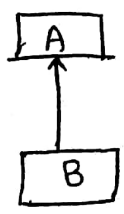
setattr(obj, attr, value)

delattr(obj, attr)

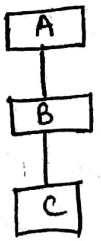
```
class Employee
    def __init__(self.name, SSN):
        self.name = name
        self.ssn = ssn
    def display(self):
        print self.name
e1 = Employee('Abc', 123)
e1.display()
print(hasattr(e1, ssn))
print(getattr(e1, ssn))
```

Inheritance :-

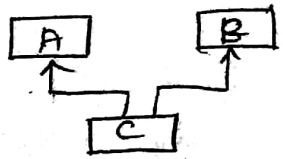
Single



Multilevel



Multiple



General Syntax for Inheritance:-

```
class Parent:
    // class defs (attributes/methods)
class child(Parent):
    //class defs. (attribute/methods).
```

Syntax for multilevel :-

```
class A:
    //defs
class B(A):
    //defs.
class C(B):
    //defs.
```

Syntax for multiple :-

```
class A:
    //defs.
class B:
    //defs
class C(A,B):
    //defs.
```

```
class Parent:
    def displayP(self)
        print "Parent Method"
class Child(Parent):
    def displayC(self):
        print("Child Method")

p1 = Parent()
p1.displayP()
c1 = Child()
c1.displayC()
c1.displayP()
p1.displayC()
```

→ throws error

super() :- Using this keyword, parent class methods or attributes can be invoked.

Eg:-

```
class Parent:
    def __init__(self):
        print("Parent Constructor")
    def displayP(self):
        print("Parent Method")
class Child(Parent):
    def __init__(self):
        print("Child Constructor")
        super()
    def displayC(self):
        print("Child Method")
P1 = Parent() //Outputs Parent Constructor.
C1 = Child() //Output: Child Constructor.
                       Parent Constructor.
```

tax for 2.7 :-

super(child, self).__init__()

Syntax in 3:- super().methodname(args)

Design a class named Fan to represent a Fan, the class contains 3 constants named slow, medium, fast with values 1, 2 and 3 to denote the fan speed. A private integer data field named speed that specifies the speed of the fan. A private boolean data field named on that specifies whether the fan is on (the default is false). A private float data field named radius that specifies the radius of the fan. A private string data field named color that specifies color of the fan. The accessor & mutator methods for all 4 datafields. A constructor that creates a fan with specified speed (default slow), radius (default 5), color (default blue) and on (default false). Write a test program that creates 2 fan objects, for the first object assign the maximum speed, radius 10, color yellow and turn it on. Assign medium speed, radius 5, color blue and turn it off for the second object. Display each objects speed, radius, color and on properties.

Exception handling in python:-

→ An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instruction.

Syntax:-
```
try:
    #write some code
    #that might throw exception
except <ExceptionType>:
    #Exception handler, alert the user.
```

Syntax:-
```
try:
    <body>
except <ExceptionType1>:
    <handler1>
except <ExceptionTypeN>:
    <handlerN>
except:
    <handlerExcept>
else:
    <process_else>      // Statements under the else clause run only when no exception is raised.
finally:
    <process_finally>   // Statements in finally block will run every time no matter exception occurs or not.
```

Raising exception :-

raise ExceptionClass("Your argument")

tkinter : module used for creating different graphical user interface.

Eg:-
```
import tkinter
window = tkinter.Tk()
#Code to add widgets will go here...
window.mainloop()
```

→ Default name of the window will be tk.

```
import tkinter
window = tkinter.Tk()
window.title("First tkinter")
window.mainloop()
```

Button :- The button widget is used to display buttons in your application.

```
w0 = Button(master, option=value, ...)
```

→ master : This represents the parent window.

→ options : Here is the list of most commonly used options for this widget. These options can be used as key-value pairs seperated by commas.

Options:-
- activebackground
- activeforeground
- bd
- bg
- command
- fg
- font

Eg:- →
```
import tkinter
window = tkinter.Tk()
button = tkinter.Button(window, text="Hello")
button.pack()
window.mainloop()
```

→
```
import tkinter
window = tkinter.Tk()
button = tkinter.Button(window, text="Hello", bd=5,
        bg="yellow", fg="red", font=20, justify="center",
                                        height=3)
button.pack()
window.mainloop()
```

Design a form with 2 buttons. Text of button 1 is Say Hi & button 2 is ~~As if when I~~ Say Hello. As if when you click the button, display the message in terminal.

```
def sayhi():
    print("Hi")
def sayhello():
    print("Hello")
```

→
```
import tkinter
window = tkinter.Tk()
button1 = tkinter.Button(window, text="Say Hi",
                         command=sayhi)
button2 = tkinter.Button(window, text="Say Hello", sayhello)
button1.pack()
button2.pack()
window.mainloop()
```

Label :- This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.

Syntax :

```
w = Label(master, option, ....)
```

Eg :-1)

```
import tkinter
window = tkinter.Tk()
label = tkinter.Label(window, text = "Hey, How are you?")
label.pack()
window.mainloop()
```

2)

```
import tkinter
window = tkinter.Tk()
label = tkinter.Label(window, text = "Hey, How are you?",
        bg = "orange", fg = "black", font = "Timesnewroman",
        bd = 4)
label.pack()
window.mainloop()
```

Entry (textbox) :-

The Entry widget is used to accept single-line text strings from a user.

Syntax :- `w = Entry(master, option, ...)`

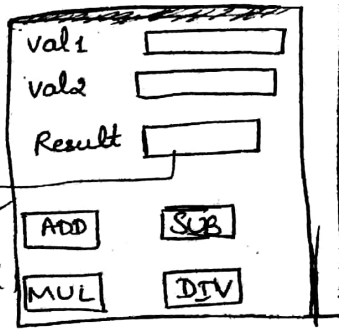
Eg :-

```
import tkinter
window = tkinter.Tk()
label = tkinter.Label(window, text = "Username")
entry = tkinter.Entry(window)
label.pack()
entry.pack()
window.mainloop()
```

Write a Tkinter program to implement simple calculator.

```
def add():
    a = var1.get()
    b = var2.get()
    var3.set(a+b)
import tkinter
window = tkinter.Tk()
label1 = tkinter.Label(window,
                       text = "val1")
label2 = tkinter.Label(window,
                       text = "val2")
entry1 = tkinter.Entry1(window, textvariable = var1)
entry2 = tkinter.Entry2(window, textvariable = var2)
label3 = tkinter.Label(window, text = "Result")
entry3 = tkinter.Entry3(window, textvariable = var3)
```

→ Ility for SUB, MUL, DIV

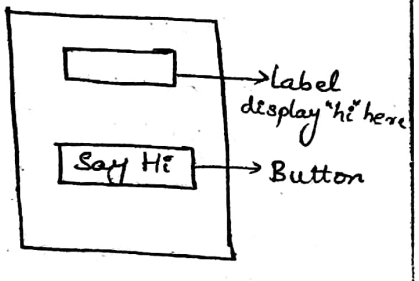


label / textfield

```
var1 = tkinter.var1()
var2 = tkinter.var2()
```

```
def display:
    l.config(text="hi")
```

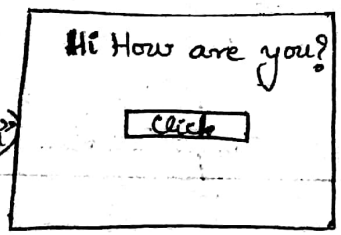
→




```
import tkinter
window = tkinter.Tk()
l = tkinter.Label(window)
b = tkinter.Button(window, text="Say Hi", command=display)
l.pack()
b.pack()
window.mainloop()
```

Design a form,

```
def display:
    l.config(text="Good")
```



As you click the Button "Click", then change the text to "Good"

→

```
import tkinter
window = tkinter.Tk()
l = tkinter.Label(window, text="Hi How are you?")
b = tkinter.Button(window, text="Click", command=display)
l.pack()
b.pack()
window.mainloop()
```

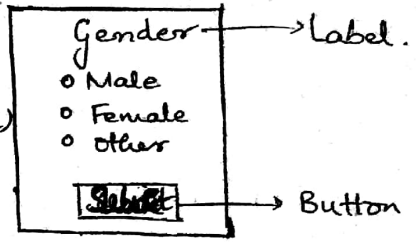
Radiobutton :-

This widget implements a multiple choice button, which is a way to offer many possible selections to the user and lets user choose only one of them.

Syntax :- w = Radiobutton(master, options, ...)

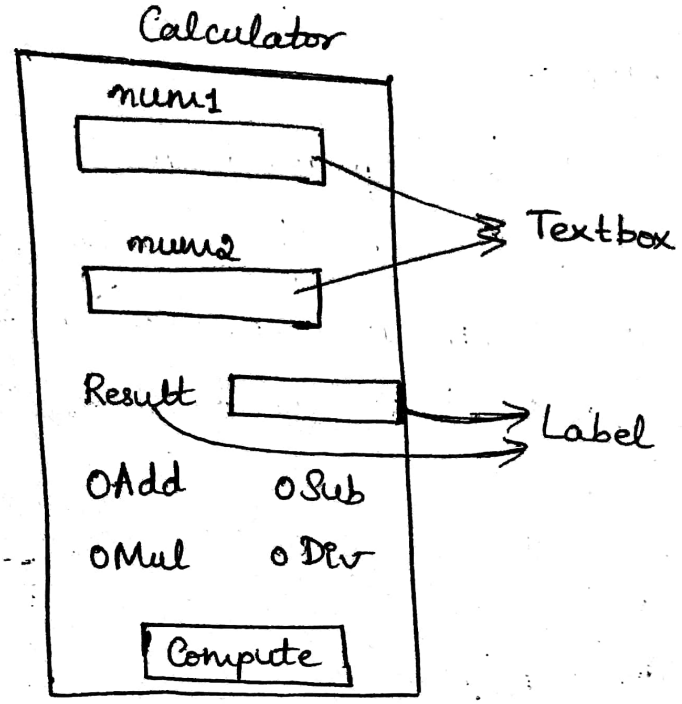
Design a form,

```
def display:
    tup = "You selected" + var.get()
    lab2.config(text=tup)
```



→

```
import tkinter
root = tkinter.Tk()
var = tkinter.StringVar()
l = tkinter.Label(root, text="Gender")
R1 = tkinter.Radiobutton(root, text="Male", variable=var, value="Male")
R2 = tkinter.Radiobutton(root, text="Female", variable=var, value="Female")
R3 = tkinter.Radiobutton(root, text="Other", variable=var, value="Others")
b = tkinter.Button(root, text="Submit", command=display)
lab2 = tkinter.Label(win)
l.pack()
R1.pack()
R2.pack()
R3.pack()
b.pack()
lab2.pack()
root.mainloop()
```




```
import tkinter
window = tkinter.Tk()
window.Title("Calculator")
l1= tkinter.Label(window, text= "num1")
l2= tkinter.Label(window, text= "num2")
var1 = tkinter.IntVar()
var2 = tkinter.IntVar()
entry1 =tkinter.Entry(window, textvariable = var1)
entry2 = tkinter.Entry(window, textvariable = var2)
```