Packing and Unpacking

Packing is process of grouping individual objects into single object.
This is done by assigning multiple values to single variable.

```
>>> A=10,20,30,40,50
>>> print(A)
(10, 20, 30, 40, 50)
>>> print(type(A))
<class 'tuple'>
>>> B="A","B","C","D","E"
print(B)
('A', 'B', 'C', 'D', 'E')
```

Unpacking is process of reading values from collection and assigning to multiple variables (OR) assigning collection object to multiple variables is called unpacking.

```
>>> L1=[10,20,30,40,50]
>>> a,b,c,d,e=L1
>>> print(a,b,c,d,e)
10 20 30 40 50
>>> s1,s2,s3,s4,s5,s6="PYTHON"
>>> print(s1,s2,s3,s4,s5)
P Y T H O
>>> print(s6)
N
>>> a,b,c,d,e=(10,20,30,40,50)
>>> print(a,b,c,d,e)
10 20 30 40 50
>>> n1,n2,n3=[10,20,30,40,50]
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    n1,n2,n3=[10,20,30,40,50]
ValueError: too many values to unpack (expected 3)
>>> n1,n2,n3,*n4=[10,20,30,40,50,60,70,80]
>>> print(n1,n2,n3,n4)
10 20 30 [40, 50, 60, 70, 80]
>>> a,*b,*c=[10,20,30,40,50,60,70]
SyntaxError: multiple starred expressions in assignment
```

```
>>> a,b,c={1:10,2:20,30:40}
>>> print(a,b,c,sep="\n")
1
2
30
>>> a,b,c={10,20,30}
>>> print(a,b,c)
10 20 30
>>> a,b,c={1:10,2:20,3:30}.values()
>>> print(a,b,c)
10 20 30
>>> a,b,c={1:10,2:20,3:30}.items()
>>> print(a,b,c,sep="\n")
(1, 10)
(2, 20)
(3, 30)
```

Functions

Python is multi paradigm programming language.
A programming paradigm defines set of rules and regulations for organizing of data and instructions.

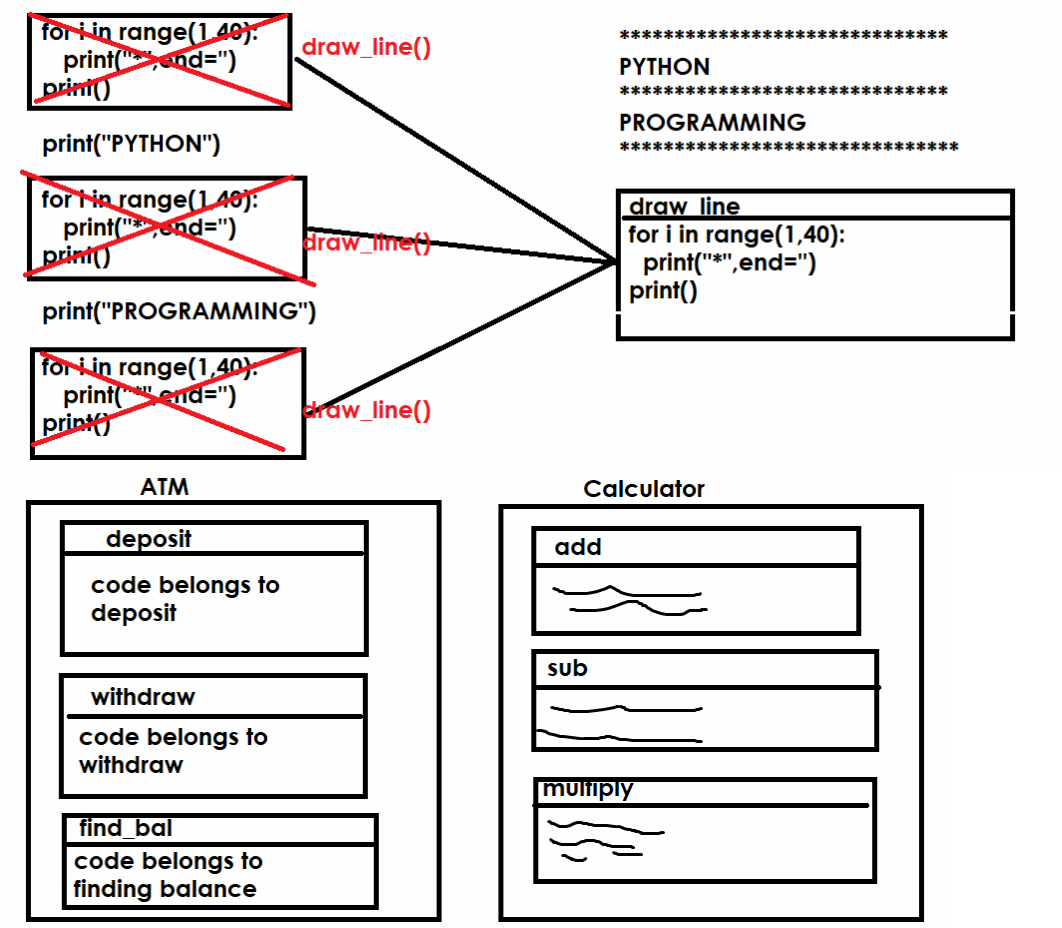
1. Procedural Oriented Programming (POP)
2. Object Oriented Programming (OOP)
3. Modular Oriented Programming (MOP)
4. Functional Oriented Programming (FOP)

Procedural Oriented Programming
In procedural oriented programming instructions organized according their operations by dividing into small pieces called sub routines (Procedures/Functions).

What is function?
A function is small program within program.
A function is self contained block, which contains set of instructions to perform operation.
A function is named block.
Functions are building blocks of procedural oriented programming.

Advantage of functions

1. **Modularity**: Dividing programming instructions according to their operations into small pieces
2. **Reusability:** It allows to code once and using many times. This avoid code redundancy
3. **Readability:** Easy to understand
4. **Efficiency:** functions are efficient, it increases efficiency of the program by avoiding code redundancy



Types of functions

Functions are 2 types

1. Predefined functions
2. User defined functions

The existing functions are called predefined functions. These are also called library functions.
Example: print(),len(),max(),min(),sum(),oct(),hex(),bin(),chr(),ord(),…

The functions build by programmer are called user defined functions. These are application specific functions.
Example: deposit(), withdraw(),login(),logout(),…

Basic steps to work with functions

1. Defining function (OR) Writing function
2. Calling function or invoking function

Defining function (OR) Writing functions
A function is defined using “def” keyword.

Syntax:

```
def function-name([parameters]):
      ‘’’doc string’’’
      statement-1
      statement-2
```

A function is defined,

1. With parameters
2. Without parameters
3. With return value
4. Without return value

Example:

```
def sayhello():
   print("Welcome to Functions")

def fun1():
   print("inside function1")
def fun2():
   print("inside function2")

sayhello()
sayhello()
fun1()
fun2()
```

Output

```
Welcome to Functions
Welcome to Functions
inside function1
inside function2
```

Function must be defined before calling or invoking

```
add()
def add():
    print("Add function")
```

Output

```
Traceback (most recent call last):
  File "D:/fspmar5pm/test236.py", line 1, in <module>
    add()
NameError: name 'add' is not defined
```

Memory is allocated or reserved or created for function, when the function is called or invoked.
Memory is deleted or function deleted from memory after execution of function.
Whenever function is called, the execution control switched from calling place to called function and after execution of called function returns to calling place.

Example:

```
def draw_line():
    for i in range(30):
        print("*",end="")
    print()


draw_line()
print("PYTHON")
draw_line()
```

Output

```
******************************
PYTHON
```

****

Local variables