

Name: Md. Rashid Aziz

Roll No: 2003013

Batch- C11

Aim : To study and work with functions in python

Theory:

► list:

Lists are used to store multiple items in a single variable. Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage. Lists are created using square brackets

► list methods:

* list.append(x)

Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

* list.extend(iterable)

Extend the list by appending all the items from the iterable. Equivalent to `a[len(a):] = iterable`.

* list.insert(i, x)

Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

* list.remove(x)

Remove the first item from the list whose value is equal to x. It raises a `ValueError` if there is no such item.

* list.pop([i])

Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the `i` in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

* `list.clear()`

Remove all items from the list. Equivalent to `del a[:]`.

* `list.index(x[, start[, end]])`

Return zero-based index in the list of the first item whose value is equal to `x`. Raises a `ValueError` if there is no such item. The optional arguments `start` and `end` are interpreted as in the slice notation and are used to limit the search to a particular subsequence of the list. The returned index is computed relative to the beginning of the full sequence rather than the `start` argument.

* `list.count(x)`

Return the number of times `x` appears in the list.

* `list.sort(*, key=None, reverse=False)`

Sort the items of the list in place (the arguments can be used for sort customization, see `sorted()` for their explanation).

* `list.reverse()`

Reverse the elements of the list in place.

* `list.copy()`

Return a shallow copy of the list. Equivalent to `a[:]`.

► list slicing:

In Python, list slicing is a common practice and it is the most used technique for programmers to solve efficient problems. Consider a python list, In-order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:) With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

Syntax:

`Lst[Initial : End : IndexJump]`

If `Lst` is a list, then the above expression returns the portion of the list from index `Initial` to index `End`, at a step size `IndexJump`.

► functions:

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. In Python a function is defined using the def keyword: To call a function, use the function name followed by parenthesis: Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma. The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name: By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less. To let a function return a value, use the return statement

► recursion:

Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result. The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

► decorators:

Decorators are a very powerful and useful tool in Python since it allows programmers to modify the behaviour of function or class. Decorators allow us to wrap another function in order to extend the behaviour of the wrapped function, without permanently modifying it. This is also called metaprogramming because a part of the program tries to modify another part of the program at compile time.

Program 1:

```
# 1
```

```
def isPerfect(n):
```

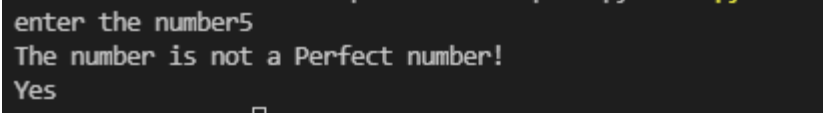
```
    sum1 = 0
```

```

for i in range(1, n):
    if(n % i == 0):
        sum1 = sum1 + i
if (sum1 == n):
    print("The number is a Perfect number!")
else:
    print("The number is not a Perfect number!")
n = int(input("enter the number"))
isPerfect(n);

```

Output:



```

enter the number5
The number is not a Perfect number!
Yes

```

Program 2:

2

```

def ispangram(str):
    alphabet = "abcdefghijklmnopqrstuvwxyz"
    for char in alphabet:
        if char not in str.lower():
            return False

    return True

```

```
string = 'th quick brown fox jumps over the lazy dog'
if(ispangram(string) == True):
    print("Yes")
else:
    print("No")
```

Program 3:

```
def add(*argv):
    print(argv)
    sum = 0;
    for i in argv:
        sum += i;
    return sum;

def mul(*argv):
    mul = 1;
    for i in argv:
        mul *= i;
    return mul;

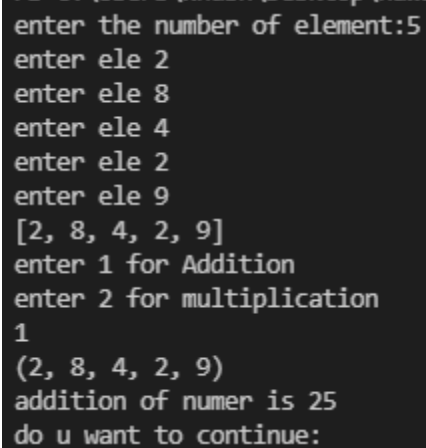
ch = 'y';
while ch == 'y':
    n = int(input("enter the number of element:"));
    arr = [];
    for i in range(0, n):
```

```

a = int(input("enter ele "));
arr.append(a);
print(arr);
print("enter 1 for Addition\nenter 2 for multiplication");
b = int(input());
if b == 1:
    print("addition of numer is", add(*arr));
else:
    print("multiplication of numbers is", mul(*arr));
ch = input("do u want to continue: ")

```

Output



```

enter the number of element:5
enter ele 2
enter ele 8
enter ele 4
enter ele 2
enter ele 9
[2, 8, 4, 2, 9]
enter 1 for Addition
enter 2 for multiplication
1
(2, 8, 4, 2, 9)
addition of numer is 25
do u want to continue:

```

Program 4:

4

```

def factorial(n):
    if n<=1:
        return n;
    return n*factorial(n-1);

```

```
n = int(input("enter the number "));  
print(factorial(n));
```

Program 5:

5

```
def square(n):  
    def increaseBy4(a):  
        return a+4;  
    return increaseBy4(n*n);  
def cube(n):  
    def mulBy2(a):  
        return a*2;  
    return mulBy2(n*n*n);
```

```
a = int(input("enter the number"));  
b = int(input("enter the 2nd number"));  
print(square(a));  
print(cube(b));
```