1) **Docstring** is a statement inside a function which describes the use of that function. These are placed inside triple quotes after defining the function and before the operations to be performed inside the function.

Syntax:

```
Def function_name():
    """ Here we enter the Docstring"""
    #operations to be performed
```

We can display the docstring of a function using the syntax,

```
print(function_name.__doc__)
```

2) **A module** is a .py file which consists of variables, functions, classes etc. Its very easy to create a module by saving the file with the extension '.py'.

To import a module in python, the following syntax is used.

```
Import module_name
```

By importing a module to a python file, we can use the variables, functions, classes etc defined in the module, in the current python file.

3) Local variable

Consider we have two variables with the same name. One is inside the function and the other is in the main body of the program. If we change the value of the variable inside the function, it will not affect the variable in the main body. Because the variable inside the function is local to that function only.

Eg:

```
x = 20
def my_func(x):
   print("The value of x is",x)
   x=100
   print("Ran the function, the value of x inside function changed to ",x)

print(my_func(x))
print("value of x outside function is",x)
```

Global variable

We can change the value of local variable by assigning it as a global variable. So if we change the value inside the function, it will be reflected in the main body also.

Eg:

```
x = 2
```

```
def my_func():
    global x
    print("x is made global inside the function")
    x = 5
    print("After calling the fn, value changed to ",x)

print("The value of x before calling the function is ",x)
my_func()
print("The value of x outside the function is",x)
```

4) Don't know the answer.
5) Generating random numbers in python is done using a module named random. Random module consists of a function called **randint().** This function returns number between two limits that is passed in to the function as arguments.

Syntax: import random
```
Random_number = random.randint(1,10)
print(Random_number)
```

6) **Functions** are useful devices that group together a block of statements those were used very often. The main advantage of function is that there is no need to type the same statements repeatedly.

Syntax: def function_name(parameters):
```
"""Docstring"""
#operations to be performed
return #returns the result of the function
```

7) ***args** is called the arbitrary arguments in python. Arbitrary arguments are created by putting a * before the parameter inside the parantheses. It is used when the programmer don't know how many arguments are to be passed in the function while calling it. By using the arbitraty arguments, it will return a tuple of values as output.

Eg:
```
def my_func(*args):
    print(args)

my_func(1,2,3,4)
```

****kwargs** is called the arbitrary keyword argument. These are created by putting ** before the parameter. When the programmer want to give input in the form of key value pair and also he doesn't know the exact number of inputs, arbitrary keyword arguments are used.

When using arbitrary keyword argument, the function will return a dictionary of key-value pair.

Eg:
```
def my_func(**kwargs):
    print(kwargs)

my_func(name='aswin',place='aluva',course='python')
```

8) Exceptions

Even if the statement and expression in a program is syntactically correct, it may raise some errors while executing it. The errors raised while executing a program is called exceptions.

These errors can be handled by using Exception handling.

Eg:
```
try:
    a = 2
    b = 0
    result = a/b
except ZeroDivisionError:
    print("A number can't be divisible by zero")
else:
    print(result)
```

9) **KeyboardInterruptError**: It's a StandardError raised when we press Ctrl+C or Ctrl+Z while executing a program.
KeyError: KeyError raises when a entered key is not found in a dictionary.
TypeError: TypeError occurs when the particular operation is not valid for the given datatype.
ValueError: It arises when a variable got a value of different datatype.

10)
```
class Exception_name(Exception):
    def __init__(self,message):
        self.message=message
```

11) **AssertionError** raises when a given expression in the assert statement fails.

Syntax: assert expression,Argument

Eg:
```
def my_func(x):
    try:
        assert x>10,"You entered a lesser value"
    except AssertionError as e:
        print(e)
    else:
```

print(x)

12) File handling operations:

- Read a file: Open a file to read the information inside the file.

  Syntax: f=open("filename.txt",r)

- Write in to a file: Open a file to write into the file. It opens a new file if the file with the given name doesn't exists or it truncates the data if the file exists.

  Syntax: f=open("filename.txt",w)

- Append in to the file: Appending means updating the date into a file. It won't overwrite the information in the file but add the given data to the end of the file.

  Syntax: f=open("fileopen.txt",a)

13) Methods to read a file

- A file can be read without mentioning the typecode inside the open() function. It opens the file and read the file by default.

  Syntax: f= open("filename.txt")

- By mentioning the typecode r inside the open() function.

  Syntax: f= open("filename.txt",r)

- Other method to open a file is using the function with open(). The advantage of this function is that you don't need to explicitly call the close() function to close the file.

  Syntax: with open("filename.txt",r) as f:
  #perform the operations

14) **Readlines()** method returns a list of each lines of the files as an element of the list.

15) re module

16)
```
def Lastelement(lst):
    if len(lst)>0:
        return lst[len(lst)-1]
    else:
        print("None")
```

17)
```
def Singlelettercount(word,letter):
    if letter in word:
        c = word.count(letter)
        print(c)
    else:
        print('0')
```

18)
```
f = open("E:/Files/alphabet.txt",'w')
f.write("abcdefghijklmnopqrstuvwxyz")
f.close()
new = open("E:/Files/threeletter.txt",'w')
f = open("E:/Files/alphabet.txt",'r')
for i in range(26//3+1):
    temp = f.read(3)
    new.write(temp)
    new.write("\n")
f.close()
new.close()
```

19)
```
Animals = ["elephant","horse","squirrel","cat","cow","rabbit"]
lst = []
for i in Animals:
    try:
        lst.append(i[4])
    except IndexError as e:
        pass
else:
    print(lst)
```

20)
```
import datetime as dt
today = dt.date.today()
yesterday = today - dt.timedelta(days=1)
tomorrow = today + dt.timedelta(days=1)
print("Yesterday's date is ",yesterday)
print("Today's date is ",today)
print("Tomorrow's date is ",tomorrow)
```