Python Comment Lines

In [ ]:
```
# ipynb stands for Interactive Python Note Book
# We are working on Anaconda Jupyter Lab
# Other Python popular editors available on Anaconda is "Jupyter Notebook" and "Spider"
# Python is a case sensative, zero base, functional, fully Object Oriented Programming (OOP) language
```

Python Introduction

In [2]:
```
print ("Hello to all...")
print ("Welcome to all...")
```

```
Hello to all...
Welcome to all...
```

In [6]:
```
print ("Hello", "and Welcome")
```

```
Hello    and Welcome
```

In [7]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Python Print Statements

In [23]:
```
print ("Hello to all...", end = ", ")
print ("Welcome to all...")
print ("Hello", "and Welcome", sep = " - ")
print ("Good Morning " * 5)
print ('Good day...')
print ("Good Morning \n" * 5)
```

```
Hello to all..., Welcome to all...
Hello - and Welcome
Good Morning Good Morning Good Morning Good Morning Good Morning
Good day...
Good Morning
Good Morning
Good Morning
Good Morning
Good Morning
```

Python Input Statements

In [26]:
```
num1 = input("Please enter the first number: ")
num2 = input("Please enter the second number: ")
print (num1, type(num1), num2, type(num2))
total = num1 + num2
print (num1, "+", num2, "=", total)
```

```
100 <class 'str'> 200 <class 'str'>
100 + 200 = 100200
```

In [27]:
```
num1 = int(input("Please enter the first number: "))
num2 = int(input("Please enter the second number: "))
print (num1, type(num1), num2, type(num2))
total = num1 + num2
print (num1, "+", num2, "=", total)
```

```
100 <class 'int'> 200 <class 'int'>
100 + 200 = 300
```

In [28]:
```
num1 = int(input("Please enter the first number: "))
num2 = int(input("Please enter the second number: "))
```

```
print (num1, type(num1), num2, type(num2))
total = num1 + num2
```

```
200 <class 'int'> 300 <class 'int'>
```

In [40]:

```
print ("So", num1, "+", num2, "=", total)
print ("So " + str(num1) + " " + " + str(num2) + " " = " + str(total))
print ("So {} + {} = {}".format(num1, num2, total))   # place holder
print ("So {0} + {1} = {2}".format(num1, num2, total))   # numbered/indexed place holder
print ("So {2} + {1} = {0}".format(total, num2, num1))   # numbered/indexed place holder
print ("So {fnum} + {snum} = {result}".format(fnum=num1, snum=num2, result=total))   # labeled place holder
print ("So {fnum} + {snum} = {result}".format(result=total, snum=num2, fnum=num1))   # labeled place holder
print ("So %d + %d = %d"%(num1, num2, total))
print ("So %d + %f = %d"%(num1, num2, total))
print ("So %d + %10.6f = %d"%(num1, num2, total))
print ("So %d + %12.5f = %d"%(num1, num2, total))
print (f"So {num1} + {num2} = {total}")
```

```
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300 = 500
So 200 + 300.000000 = 500
So 200 + 300.000000 = 500
So 200 +    300.00000 = 500
So 200 + 300 = 500
```

Python Operators

In [44]:

```
# Arithmetic Operators: + - * / // % **
print (100 + 20)   # addition operation
print (100 - 20)   # subtraction operation
print (100 * 20)   # multiplication operation
print (100 / 30)   # float division operation
print (100 // 30)  # integer division operation
print (100 % 40)   # modulus operation
print (100 ** 2)   # exponentiation operation
```

```
120
80
2000
3.3333333333333335
3
20
10000
```

In [48]:

```
# Logical Operators: and, or, not
print (False and False, False and True, True and False, True and True)
print (False or False, False or True, True or False, True or True)
print (not(False or False), not(True and True))
```

```
False False False True
False True True True
True False
```

In [52]:

```
# relational Operators: > >= < <= != ==
print (100 > 80, 100 >= 80, 100 < 500, 100 <= 500, 100 != 80, 100 == 100)
```

```
True True True True True True
```

In [55]:

```
# Ternary Operator: Unary (one operand) -10 +20, Binary (two operands) 10+20, Ternary (three operands)
num = 100
result = "EVEN" if (num % 2  == 0) else "ODD"  # three operands: True Part, Condition and False Part
print (result)

num = 101
result = "EVEN" if (num % 2  == 0) else "ODD"
print (result)
```

```
EVEN
ODD
```

ASCII (American Standard Code for Information Interchange) Code: ---------- ASCII Code are of 8 bits: Value range is 0 to 255 Category-1: 0 to 127 (Normal ASCII codes, printable) A (65), B (66), ..., Z (90), a (97), b (98), ..., z (122), 0 (48), 1 (49), ..., 9 (57), Enter (13), Esc (27), Tab (9), BS (8), Space (32) Category-2: 128 to 255, Non-printible extended ASCII Codes used for control characters

In [59]:

```
print (ord("A"), ord("Z"), ord("a"), ord("z"))
print (chr(65), chr(90), chr(97), chr(122))
```

```
65 90 97 122
A Z a z
```

In [63]:

```
# Bitwise Operator: & | ^ ~ (1's complement)
#   65 (A) => 64 + 1      = 0100 0001             97 (a) => 64 + 32 + 1 = 0110 0001
#                    or (|) 0010 0000 (32)                        and (&) 1101 1111 (223)
#                           ---------                                    ---------
#   97 (a) => 64 + 32 + 1 = 0110 0001             65 (A) => 64 + 1      = 0100 0001
mychar = "A"
print (mychar, chr(ord(mychar) | 32))
mychar = "a"
print (mychar, chr(ord(mychar) & 223))
mychar = "a"
print (mychar, chr(ord(mychar) & (~32)))
```

```
A a
a A
a A
```

Python Conditional Statements

In [78]:

```
# conditional statements
# find out the maximum of three user given numbers
num1 = int(input("Please enter the first number: "))
num2 = int(input("Please enter the second number: "))
num3 = int(input("Please enter the third number: "))
if (num1 > num2):
    if (num1 > num3):
        print ("So the first number is the maximum number...")
        print (f"So the maximum number is {num1}...")
    else:
        print ("So the third number is the maximum number...")
        print (f"So the maximum number is {num3}...")
elif (num2 > num3):
    print ("So the second number is the maximum number...")
    print (f"So the maximum number is {num2}...")
else:
    print ("So the third number is the maximum number...")
    print (f"So the maximum number is {num3}...")
print ("End of the program...")
```

```
So the third number is the maximum number...
So the maximum number is 33...
End of the program...
```

In [80]:

```
# find out the maximum of three user given numbers
num1 = int(input("Please enter the first number: "))
num2 = int(input("Please enter the second number: "))
num3 = int(input("Please enter the third number: "))
if (num1 > num2 and num1 > num3):
        print ("So the first number is the maximum number...")
        print (f"So the maximum number is {num1}...")
elif (num2 > num3):
    print ("So the second number is the maximum number...")
    print (f"So the maximum number is {num2}...")
else:
    print ("So the third number is the maximum number...")
    print (f"So the maximum number is {num3}...")
print ("End of the program...")
```

```
So the second number is the maximum number...
So the maximum number is 88...
End of the program...
```

In [82]:

```
# find out whether a given number is EVEN or ODD
num1 = int(input("Please enter the number: "))
if (num1 % 2 == 0):
    print ("So the number is the EVEN number...")
    print ("Then part is executed...")
else:
    print ("So the number is the ODD number...")
    print ("Else part is executed...")
print ("End of the program...")
```

```
So the number is the ODD number...
Else part is executed...
End of the program...
```

In [87]:

```
# find out whether a given number is EVEN or ODD
```

```
num1 = int(input("Please enter the number: "))
if (num1 % 2 == 0): print ("So the number is the EVEN number..."); print ("Then part is executed...")
else: print ("So the number is the ODD number..."); print ("Else part is executed...")
print ("End of the program...")
```

```
So the number is the EVEN number...
Then part is executed...
End of the program...
```

In [42]:

```
# find out whether a given number is EVEN or ODD
num1 = int(input("Please enter the number: "))
if (num1 % 2 == 0):
    pass    # pass is a statement placeholder and a keyword
else:
    print ("So the number is the ODD number...")
    print ("Else part is executed...")
print ("End of the program...")
```

```
So the number is the ODD number...
Else part is executed...
End of the program...
```

Python Loop Statements

In [92]:

```
# Looping with For Loop...
for i in range(5):
    print ("i =", i)
print ("")
for i in range(0, 5, 1):
    print ("i =", i)
```

```
i = 0
i = 1
i = 2
i = 3
i = 4

i = 0
i = 1
i = 2
i = 3
i = 4
```

In [111…

```
count = 0
for i in range(-10, 10, 1):
    print ("i =", i, end = ", ")
    count += 1
print (f"\nSo number of iterations = {count}")

print ("")

count = 0
for i in range(10, -10, -1):
    print ("i =", i, end = ", ")
    count += 1
else:
    print ("\n\nElse part is executing...")
print (f"\nSo number of iterations = {count}")
```

```
i = -10, i = -9, i = -8, i = -7, i = -6, i = -5, i = -4, i = -3, i = -2, i = -1, i = 0, i = 1, i = 2, i = 3, i = 4,
i = 5, i = 6, i = 7, i = 8, i = 9,
So number of iterations = 20

i = 10, i = 9, i = 8, i = 7, i = 6, i = 5, i = 4, i = 3, i = 2, i = 1, i = 0, i = -1, i = -2, i = -3, i = -4, i = -
5, i = -6, i = -7, i = -8, i = -9,

Else part is executing...

So number of iterations = 20
```

In [97]:

```
for i in [100, 300, 400, 500]:
    print(f"i = {i}")
```

```
i = 100
i = 300
i = 400
i = 500
```

In [102…

```
for i in range(10):
    if (i == 8):
        print ("Breaking out for i =", i)
```

```
        break
    print (f"{i}, ")
print ("End of the program...")
```

```
0,
1,
2,
3,
4,
5,
6,
7,
Breaking out for i = 8
End of the program...
```

In [104…

```
for i in range(10):
    if (i == 4 or i == 8):
        print ("Continuing with the next iteration...")
        continue
    print (f"i = {i}, ")
print ("End of the program...")
```

```
i = 0,
i = 1,
i = 2,
i = 3,
Continuing with the next iteration...
i = 5,
i = 6,
i = 7,
Continuing with the next iteration...
i = 9,
End of the program...
```

In [107…

```
print (range(10), list(range(10)))
```

```
range(0, 10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [113…

```
for i in range(10):
    if (i == 8):
        print ("Breaking out for i =", i)
        break
    print (f"{i}, ")
else:
    print ("Else block is executing...")
print ("End of the program...")
```

```
0,
1,
2,
3,
4,
5,
6,
7,
Breaking out for i = 8
End of the program...
```

In [109…

```
for i in range(10):
    if (i == 4 or i == 8):
        print ("Continuing with the next iteration...")
        continue
    print (f"i = {i}, ")
else:
    print ("Else block is executing...")
print ("End of the program...")
```

```
i = 0,
i = 1,
i = 2,
i = 3,
Continuing with the next iteration...
i = 5,
i = 6,
i = 7,
Continuing with the next iteration...
i = 9,
Else block is executing...
End of the program...
```

In [120…

```
num = int(input("Please enter one integer: "))
for i in range(2, int(num ** 0.5) + 1):
    if (num % i == 0):
        print (f"{num} is NOT a Prime Number...")
```

```
        break
else:
    print (f"{num} is a Prime Number...")
print ("End of the program...")
```

```
101 is a Prime Number...
End of the program...
```

In [119…

```
num = 101
print (num, num ** 0.5, int(num ** 0.5))
```

```
101 10.04987562112089 10
```

In [121…

```
print ("Visit Doctor...")
for day in range(1, 6):
    print (f"Day No: {day}, Good morning...")
    for medi in range(1, 4):
        print (f"Day No: {day} and Medicine No: {medi}...")
    print (f"Day No: {day}, Good night...")
    print ("------------------------------------------------------------")
else:
    print ("Thanks to Dcotor...")
```

```
Visit Doctor...
Day No: 1, Good morning...
Day No: 1 and Medicine No: 1...
Day No: 1 and Medicine No: 2...
Day No: 1 and Medicine No: 3...
Day No: 1, Good night...
------------------------------------------------------------
Day No: 2, Good morning...
Day No: 2 and Medicine No: 1...
Day No: 2 and Medicine No: 2...
Day No: 2 and Medicine No: 3...
Day No: 2, Good night...
------------------------------------------------------------
Day No: 3, Good morning...
Day No: 3 and Medicine No: 1...
Day No: 3 and Medicine No: 2...
Day No: 3 and Medicine No: 3...
Day No: 3, Good night...
------------------------------------------------------------
Day No: 4, Good morning...
Day No: 4 and Medicine No: 1...
Day No: 4 and Medicine No: 2...
Day No: 4 and Medicine No: 3...
Day No: 4, Good night...
------------------------------------------------------------
Day No: 5, Good morning...
Day No: 5 and Medicine No: 1...
Day No: 5 and Medicine No: 2...
Day No: 5 and Medicine No: 3...
Day No: 5, Good night...
------------------------------------------------------------
Thanks to Dcotor...
```

In [ ]:

```
print ("Visit Doctor...")
day = 1
while (day <= 5):
    print (f"Day No: {day}, Good morning...")
    medi = 1
    while (medi <= 3):
        print (f"Day No: {day} and Medicine No: {medi}...")
        medi += 1
    print (f"Day No: {day}, Good night...")
    print ("------------------------------------------------------------")
    day += 1
else:
    print ("Thanks to Dcotor...")
```

In [1]:

```
# Pattern printing - 1

# n = 6         i   .   *
#             -------------
# .....*        1   5   1
# ....***       2   4   3   . => (n - i)
# ...*****      3   3   5
# ..*******     4   2   7   * => (2 * i - 1)
# .*********    5   1   9
# ***********   6   0  11
#             -------------
#             Tracing Table
#
```

```
n = int(input("Please enter the number of layers: "))
i = 1
while ( i <= n ):
    blank = n - i
    star = 2 * i - 1
    print ("." * blank + "*" * star)
    i += 1

print ("\nEnd of the program...")
```

```
.........*
........***
.......*****
......*******
.....*********
....***********
...*************
..***************
.*****************
*******************

End of the program...
```

In [2]:

```
n = int(input("Please enter the number of layers: "))
i = 1
while ( i <= n ):
    print ("." * (n - i) + "*" * (2 * i - 1))
    i += 1

print ("\nEnd of the program...")
```

```
.........*
........***
.......*****
......*******
.....*********
....***********
...*************
..***************
.*****************
*******************

End of the program...
```

In [6]:

```
# Pattern printing - 2 (Using for in Python)

# n = 6         i   .   *
#             -------------
# ***********   1   0  11
# .*********    2   1   9  . => ( i - 1 )
# ..*******     3   2   7
# ...*****      4   3   5  * => ( 2 * (n - i) + 1 )
# ....***       5   4   3
# .....*        6   5   1
#             -------------
#             Tracing Table
#
n = int(input("Please enter the number of layers: "))
for i in range(1, n + 1):
    blank = (i - 1)
    star = (2 * (n - i) + 1)
    print ("." * blank + "*" * star)

print ("\nEnd of the program...")
```

```
*********************
.*******************
..*****************
...***************
....*************
.....***********
......*********
.......*******
........*****
.........***
..........*

End of the program...
```

In [12]:

```
# Pattern printing - 3 (Using for/while in Python)

# n = 11 (OOD number of layers)
# m = 6 = (n + 1) / 2
```

```
#              i   .   *
#            -------------
# .....*       1   5   1       (n, i, m)
# ....***      2   4   3  . => (m - i)
# ...*****     3   3   5
# ..*******    4   2   7  * => (2 * i - 1)
# .*********   5   1   9
# *********** __6___0__11__
# .*********   7   1   9
# ..*******    8   2   7  . => (i - m)
# ...*****     9   3   5
# ....***     10   4   3  * => 2 * (n - i) + 1
# .....*      11   5   1
#            -------------
#            Tracing Table
#
while [ True ]:
    n = int(input("Please enter the ODD number of layers: "))
    if ( n % 2 == 1 ):
        break
m = (n + 1) // 2
for i in range(1, n + 1):
    if ( i > m ): blank = (i - m); star = (2 * (n - i) + 1)
    else: blank = (m - i); star = (2 * i - 1)
    print ("." * blank + "*" * star)

print ("\nEnd of the program...")
```

```
Please enter the ODD number of layers: 11
.....*
....***
...*****
..*******
.*********
***********
.*********
..*******
...*****
....***
.....*

End of the program...
```

In [28]:

```
n = int(input("Please enter the number of layers: "))
i = 1
while ( i <= n ):
    blank= i - 1
    j = 1
    while ( j <= blank ):
        print (".", end = "")
        j = j + 1
    star = (2 * (n - i) + 1)
    j = 1
    while ( j <= star ):
        print ("*", end = "")
        j = j + 1
    i = i + 1
    print ("")
print ("\nEnd of the program...")
```

```
Please enter the number of layers: 19
*******************
.*****************
..***************
...*************
....***********
.....*********
......*******
.......*****
........***
.........*

End of the program...
```

Python Variables and IDs

In [43]:

```
num1 = -5
num2 = num1
print (num1, type(num1), id(num1))
print (num2, type(num2), id(num2))
num3 = -5
print (num3, type(num3), id(num3))
```

```
-5 <class 'int'> 140705845159536
```

```
-5 <class 'int'> 140705845159536
-5 <class 'int'> 140705845159536
```

In [34]:

```
num1 = -6
num2 = num1
print (num1, type(num1), id(num1))
print (num2, type(num2), id(num2))
num3 = -6
print (num3, type(num3), id(num3))
```

```
-6 <class 'int'> 2230715895696
-6 <class 'int'> 2230715895696
-6 <class 'int'> 2230715895632
```

In [36]:

```
# Any variable initialized with any value ranging from -5 to 256 (inclusive of the limit values)
# will have the same ids, indicating that they will occupie the same memory locations.
num1 = 257
num2 = num1
print (num1, type(num1), id(num1))
print (num2, type(num2), id(num2))
num3 = 257
print (num3, type(num3), id(num3))
```

```
257 <class 'int'> 2230715895248
257 <class 'int'> 2230715895248
257 <class 'int'> 2230715895472
```

Introduction to Python Collections

In [41]:

```
# Python Collections
# implicit objects
# List is mutable, i.e. insert, delete and update operations can be carried out on this object
# it is collection of ordered elements of same or different types of data
list1 = [1011, "Amitava", 34, "Developer", True]
print (list1, len(list1), type(list1), id(list1))

# Tuple is immutable, i.e. insert, delete and update operations can not be carried out on this object
# it is collection of ordered elements of same or different types of data
tuple1 = (1011, "Amitava", 34, "Developer", True)
print (tuple1, len(tuple1), type(tuple1), id(tuple1))

# Dictionary is mutable, i.e. insert, delete and update operations can be carried out on this object
# it is a collection of key-value pairs
dict1 = {"empid":1011, "empname":"Amitava", "empage":34, "empdesig":"Developer", "married":True}
print (dict1, len(dict1), type(dict1), id(dict1))

# Set is mutable, i.e. insert, delete and update operations can be carried out on this object
# it is unordered unique collection of elements of same or different types of data
set1 = {1011, 34, "Developer", True, "Amitava", 34, "Developer", True}
print (set1, len(set1), type(set1), id(set1))

# Frozen-Set is immutable, i.e. insert, delete and update operations can not be carried out on this object
# it is unordered unique collection of elements of same or different types of data
frzset1 = frozenset([1011, 34, "Developer", True, "Amitava", 34, "Developer", True])
print (frzset1, len(frzset1), type(frzset1), id(frzset1))
```

```
[1011, 'Amitava', 34, 'Developer', True] 5 <class 'list'> 2230733391552
(1011, 'Amitava', 34, 'Developer', True) 5 <class 'tuple'> 2230716433008
{'empid': 1011, 'empname': 'Amitava', 'empage': 34, 'empdesig': 'Developer', 'married': True} 5 <class 'dict'> 22307
32967744
{True, 34, 'Amitava', 'Developer', 1011} 5 <class 'set'> 2230736056832
frozenset({True, 34, 'Amitava', 'Developer', 1011}) 5 <class 'frozenset'> 2230736056608
```

Python User Defined Functions (UDF)

In [47]:

```
def funct1():
    print ("Hello " * 3)

funct1()
funct1()
funct1()
funct1()
print (type(funct1), id(funct1))
```

```
Hello Hello Hello
Hello Hello Hello
Hello Hello Hello
Hello Hello Hello
<class 'function'> 2230746350208
```

In [50]:

```
def funct2(msg, times):
    print (msg * times)
```

```
funct2("Welcome ", 6)
funct2("Good bye !!! ", 5)
funct2("Good Day... ", 3)
```

```
Welcome Welcome Welcome Welcome Welcome Welcome
Good bye !!! Good bye !!! Good bye !!! Good bye !!! Good bye !!!
Good Day... Good Day... Good Day...
```

In [52]:

```
def funct3(msg, times):         # here msg and times are positional parameters
    return msg * times

result = funct3("Welcome ", 5)  # here "Welcome" and 5 are positional arguments
print (result)
print (funct3("Good Bye ", 3))
print (funct3("Nice Day... ", 4))
```

```
Welcome Welcome Welcome Welcome Welcome
Good Bye Good Bye Good Bye
Nice Day... Nice Day... Nice Day... Nice Day...
```

In [61]:

```
# function with default arguments
def funct4(par1 = 111, par2 = 222, par3 = 333):
    print (f"par1 = {par1}, par2 = {par2} and par3 = {par3}...")

funct4(100, 200, 300)
funct4(100, 200)
funct4(100)
funct4()
funct4(par1 = 100, par3 = 300)
funct4(par3 = 300, par2 = 200)
```

```
par1 = 100, par2 = 200 and par3 = 300...
par1 = 100, par2 = 200 and par3 = 333...
par1 = 100, par2 = 222 and par3 = 333...
par1 = 111, par2 = 222 and par3 = 333...
par1 = 100, par2 = 222 and par3 = 300...
par1 = 111, par2 = 200 and par3 = 300...
```

In [64]:

```
def funct5(par1, par2 = None):   # three keywords in Python starts with capital letters: True, False, None
    # if (par2 == None):
    if (par2 is None):
        return par1 + par1
    else:
        return par1 + par2

print (funct5(100))
print (funct5(100, 900))
```

```
200
1000
```

In [67]:

```
def funct6(num1, num2):
    total = num1 + num2
    difference = num1 - num2
    product = num1 * num2
    quotient = num1 / num2
    remainder = num1 % num2
    return total, difference, product, quotient, remainder

tot, dif, pro, quo, rem = funct6(100, 40)
print (f"Total = {tot}, Difference = {dif}, Product = {pro}, Quotient = {quo}, Remainder = {rem}...")

result = funct6(100, 40)
print (result, len(result), type(result), id(result))
print (f"Total = {result[0]}, Difference = {result[1]}, Product = {result[2]}, Quotient = {result[3]}, Remainder = {
```

```
Total = 140, Difference = 60, Product = 4000, Quotient = 2.5, Remainder = 20...
(140, 60, 4000, 2.5, 20) 5 <class 'tuple'> 2230732269984
Total = 140, Difference = 60, Product = 4000, Quotient = 2.5, Remainder = 20...
```

In [72]:

```
# Python supports variant data type
data = 100
print (data, type(data), id(data))
data = 105.6
print (data, type(data), id(data))
data = "100"
print (data, type(data), id(data))
data = True
print (data, type(data), id(data))
print ("")
strdata = "India"
```

```
print (strdata, len(strdata), type(strdata), id(strdata))
strdata = "Japan"
print (strdata, len(strdata), type(strdata), id(strdata))
```

```
100 <class 'int'> 140705845162896
105.6 <class 'float'> 2230746735536
100 <class 'str'> 2230735886832
True <class 'bool'> 140705844877136

India 5 <class 'str'> 2230734786992
Japan 5 <class 'str'> 2230734787312
```

Class Assignment

In [77]:

```
# Shell script on a series problem
#         1   2   3   4   5    6   7   8   9   10   11  12
# total = 1 + 2 + 3 + 4 + 10 + 5 + 6 + 7 + 8 + 26 + 9 + 10 + ... n terms
# What is the 99th term of the series? Ordinary term => 80
# 5 x 20 = 100 -> 80 ordinary terms and 20 sum terms
#
# ALGORITHM:
# input n
# tsum = 0; fsum = 0; term = 1
# for i = 1 to n step 1
#       if (i % 5 == 0) then
#               fsum = fsum + tsum
#               tsum = 0
#       else
#               fsum = fsum + term
#               tsum = tsum + term
#               term = term + 1
#   end if
# end for
# print fsum
# end

n = int(input("Please enter the numbe of terms: "))
tsum=0
fsum=0
term=1
for i in range(1, n + 1):
    if ( i % 5 == 0 ):
        fsum = (fsum + tsum)
        print (f"Adding the temporary sum {tsum}...")
        tsum=0
    else:
        fsum = (fsum + term)
        tsum = (tsum + term)
        print (f"Adding the term {term}...")
        term = (term + 1)
print (f"\nSo the sum of the series is {fsum}...")
print ("\n End of the program...")
```

```
Adding the term 1...
Adding the term 2...
Adding the term 3...
Adding the term 4...
Adding the temporary sum 10...
Adding the term 5...
Adding the term 6...
Adding the term 7...
Adding the term 8...
Adding the temporary sum 26...
Adding the term 9...
Adding the term 10...

So the sum of the series is 91...

 End of the program...
```

Recursive and Non-Recursive Functions

In [85]:

```
# non-recursive function to calculate factorial of a user given number...
def factorial_nr(num):
    if (num == 0 or num == 1): return 1
    fact = num
    for i in range(2, num):
        fact = fact * i
        print (f"For i = {i} fact = {fact}...")
    return fact

n = 5
result = factorial_nr(n)
```

```
print (f"For n = {n}, factorial = {result}")
print ("")
n = 6
result = factorial_nr(n)
print (f"For n = {n}, factorial = {result}")
```

```
For i = 2 fact = 10...
For i = 3 fact = 30...
For i = 4 fact = 120...
For n = 5, factorial = 120

For i = 2 fact = 12...
For i = 3 fact = 36...
For i = 4 fact = 144...
For i = 5 fact = 720...
For n = 6, factorial = 720
```

In [2]:
```
# recursive function to calculate factorial of a user given number...
def factorial_r(num):
    if (num == 0 or num == 1): return 1  # base case, for certain inputs outputs are pre-known to us
    return num * factorial_r(num - 1)    # recursive case, here the function will call itself

n = 5
result = factorial_r(n)
print (f"For n = {n}, factorial = {result}")
print ("")
n = 6
result = factorial_r(n)
print (f"For n = {n}, factorial = {result}")
```

```
For n = 5, factorial = 120

For n = 6, factorial = 720
```

In [8]:
```
x = 100
```

In [9]:
```
print (x)
```

```
100
```

In [6]:
```
x = 20
```

Lambda Function or Anonymous Function

In [14]:
```
mysquare = lambda num: num * num

result = mysquare(5)
print (result, mysquare)
print (mysquare(6))
print (mysquare(7))
print (type(mysquare), id(mysquare), mysquare)
```

```
25 <function <lambda> at 0x00000202498F9430>
36
49
<class 'function'> 2208847336496 <function <lambda> at 0x00000202498F9430>
```

In [ ]:
```
total = (num1 + num2)     # assignment statement as it is containing one assignment operator
                          # here (num1 + num2) is an expression as it returns a value
```

In [11]:
```
myaddition = lambda num1, num2: num1 + num2

print (myaddition(5, 10))
print (myaddition(66, 33))
print (myaddition(700, 200))
print (type(myaddition), id(myaddition))
```

```
15
99
900
<class 'function'> 2208819042048
```

In [16]:
```
# recursion with lambda function
myfactorial = lambda num: 1 if (num == 0 or num == 1) else num * myfactorial(num - 1)

fact = myfactorial(5)
print (fact)
print (myfactorial(4))
```

```
print (myfactorial(6))
print (type(myfactorial), id(myfactorial), myfactorial)
```

```
120
24
720
<class 'function'> 2208846905408 <function <lambda> at 0x0000020249890040>
```

In [20]:

```
# example of a function which returns a function variable
def funct7(num):
    myproduct = lambda x: x * num
    return myproduct

var10 = funct7(10)
var20 = funct7(20)
print (var10(3))
print (var20(4))
print (funct7(30)(5))
print (var10, type(var10), id(var10))
print (var20, type(var20), id(var20))
```

```
30
80
150
<function funct7.<locals>.<lambda> at 0x00000202498F9700> <class 'function'> 2208847337216
<function funct7.<locals>.<lambda> at 0x00000202498F95E0> <class 'function'> 2208847336928
```

More on Python User Defined Functions

In [21]:

```
i = 10
def funct8():
    i = 100
    print ("Print from the function:", i, type(i), id(i))

print ("Print before calling the function:", i, type(i), id(i))
funct8()
print ("Print after calling the function:", i, type(i), id(i))
```

```
Print before calling the function: 10 <class 'int'> 140705845160016
Print from the function: 100 <class 'int'> 140705845162896
Print after calling the function: 10 <class 'int'> 140705845160016
```

In [22]:

```
i = 10
def funct8():
    global i
    i = 100
    print ("Print from the function:", i, type(i), id(i))

print ("Print before calling the function:", i, type(i), id(i))
funct8()
print ("Print after calling the function:", i, type(i), id(i))
```

```
Print before calling the function: 10 <class 'int'> 140705845160016
Print from the function: 100 <class 'int'> 140705845162896
Print after calling the function: 100 <class 'int'> 140705845162896
```

In [24]:

```
def funct8():
    global i
    i = 100
    print ("Print from the function:", i, type(i), id(i))

print ("Print before calling the function:", i, type(i), id(i))
funct8()
print ("Print after calling the function:", i, type(i), id(i))
i = 10
```

```
Print before calling the function: 100 <class 'int'> 140705845162896
Print from the function: 100 <class 'int'> 140705845162896
Print after calling the function: 100 <class 'int'> 140705845162896
```

In [29]:

```
# function with variable number of arguments
def funct9(*arg):    # *arg is defining the variable arg forcefully of a tuple type
    print (arg, len(arg), type(arg), id(arg))

funct9(101, "Anup", "Tester")
funct9(101, "Anup", "Tester", "Pune", 50000)
funct9(101, "Anup", "Tester", "Pune", 50000, "M.Tech", True)
funct9(101, "Anup", "Tester", "Pune", 50000, "M.Tech", True, "SCRUM Certified")
```

```
(101, 'Anup', 'Tester') 3 <class 'tuple'> 2208838857664
(101, 'Anup', 'Tester', 'Pune', 50000) 5 <class 'tuple'> 2208835161440
```

```
(101, 'Anup', 'Tester', 'Pune', 50000, 'M.Tech', True) 7 <class 'tuple'> 2208845510784
(101, 'Anup', 'Tester', 'Pune', 50000, 'M.Tech', True, 'SCRUM Certified') 8 <class 'tuple'> 2208835919936
```

In [30]:

```
# function with variable number of arguments
def funct10(**kwarg):    # *kwarg is defining the variable kwarg forcefully of a dictionary type
    print (kwarg, len(kwarg), type(kwarg), id(kwarg))   # here kwarg stands for keyword argument

funct10(empid=101, empname="Anup", empdesig="Tester")
funct10(empid=101, empname="Anup", empdesig="Tester", emploc="Pune", empsal=50000)
funct10(empid=101, empname="Anup", empdesig="Tester", emploc="Pune", empsal=50000, empstatus=True)
```

```
{'empid': 101, 'empname': 'Anup', 'empdesig': 'Tester'} 3 <class 'dict'> 2208838818432
{'empid': 101, 'empname': 'Anup', 'empdesig': 'Tester', 'emploc': 'Pune', 'empsal': 50000} 5 <class 'dict'> 22088392
00640
{'empid': 101, 'empname': 'Anup', 'empdesig': 'Tester', 'emploc': 'Pune', 'empsal': 50000, 'empstatus': True} 6 <cla
ss 'dict'> 2208839226432
```

In [33]:

```
# function with variable number of arguments
def funct11(*arg, **kwarg):
    print (arg, len(arg), type(arg), id(arg))
    print (kwarg, len(kwarg), type(kwarg), id(kwarg))

funct11(101, "Anup", "Tester")
print ()
funct11(emploc="Pune", empsal=50000, empstatus=True)
print ()
funct11(101, "Anup", "Tester", emploc="Pune", empsal=50000, empstatus=True)
```

```
(101, 'Anup', 'Tester') 3 <class 'tuple'> 2208819408384
{} 0 <class 'dict'> 2208839388928

() 0 <class 'tuple'> 2208744210496
{'emploc': 'Pune', 'empsal': 50000, 'empstatus': True} 3 <class 'dict'> 2208839389056

(101, 'Anup', 'Tester') 3 <class 'tuple'> 2208838306880
{'emploc': 'Pune', 'empsal': 50000, 'empstatus': True} 3 <class 'dict'> 2208839388480
```

In [ ]: