Function

In [1]:
```
#lets create simple function with name iseven, which will look whether given number is even or odd
```

In [2]:
```
def iseven_or_odd(num):
    if num%2==0:
        print("provided number is even")
    else:
        print("its a odd")

```

In [3]:
```
#calling a function
iseven_or_odd(7)
```

```
its a odd
```

In [6]:
```
#We can use return instead of print that is another way
def iseven_or_odd(num):
    if num%2==0:
        return 'even'
    else:
        return 'odd'
```

In [7]:
```
iseven_or_odd(12)
```

Out[7]:
```
'even'
```

""" we can use any methods or way inside function like list comprehension too, lets create one function which says its a capital or samll letter """

In [10]:
```
def capitalorsmall(i):
    if i==i.upper():
        return 'capital'
    else:
        return 'small'

#first of all lets check will it work?
```

In [11]:
```
capitalorsmall("ram")
```

Out[11]:
```
'small'
```

In [24]:
```
#can we do it in list comprehension way?
def capitalorsmall(L):

    capital=[i for i in L if i==i.upper()]
    small=[i for i in L if i==i.lower()]
    return capital, small
```

In [25]:
```
capitalorsmall("Nepal")
```

Out[25]:
```
(['N'], ['e', 'p', 'a', 'l'])
```

Types of Argument in Function

In [26]:
```
#But, firstly, lets understand what is parameter vs argument?
#example def capitalorsmall(L)
#what is L: It is parameter, when we create function it is parameter, but when user give value to that L, it is
```

1. Default Arguments
2. Positional Arguments
3. Keyword Arguments

In [27]:
```
#lets see small example for default argument
def power(a,b):   #here according to function there is 2 parameter
    result=a**b
    return result
```

In [28]:
```
power(2,3)
```

Out[28]:
```
8
```

In [29]:
```
#This works but think, if user forgor to pass 1 argument? it will give error
power(2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 #This works but think, if user forgor to pass 1 argument? it will give error
----> 2 power(2)

TypeError: power() missing 1 required positional argument: 'b'
```

In [30]:
```python
#so instead of this type of error message,we can use default argument
def power(a=1,b=1):#here according to function there is 2 parameter, if user forgot to pass argument then defau
    result=a**b
    return result
```

In [31]:
```python
power(3)
```

Out[31]:
```
3
```

Args and Kwargs

By using this we can provide as many input as we want in function

In [32]:
```python
#suppose lets see basic example, we did not know how many arguments/inputs will user pass, but output should be
#multiplication of all inputs
```

In [33]:
```python
def product(*num):
    product=1

    for i in num:
        product=product*i

    return product
```

In [34]:
```python
product(1,2,3,4,5,6,5)
```

Out[34]:
```
3600
```

In [35]:
```python
#how this function working, num we are giving input, it will take as tuples and multiply it, want to see?

def product(*num):
    product=1

    for i in num:
        product=product*i
    print(num)
    return product
```

In [37]:
```python
product(1,2,2,4,3,10,5,6,78,12)
```

```
(1, 2, 2, 4, 3, 10, 5, 6, 78, 12)
```

Out[37]:
```
13478400
```

In [38]:
```python
#see when we print(num), it gave use tuple.
```

In [39]:
```python
#**kwargs
```

In [43]:
```python
def display(**dict):
    for i,j in dict.items():
        print(i,"=",j)
```

In [44]:
```python
display(India="Mumbai",Nepal="Hetauda",Pakistan="Islamabad")
```

```
India = Mumbai
Nepal = Hetauda
Pakistan = Islamabad
```

In [45]:
```python
#Note: *args is storing inputs as tuples to provide output, similarly,**kwargs is storing as dictionary.
```

If someone ask you, lifetime of any particular function?

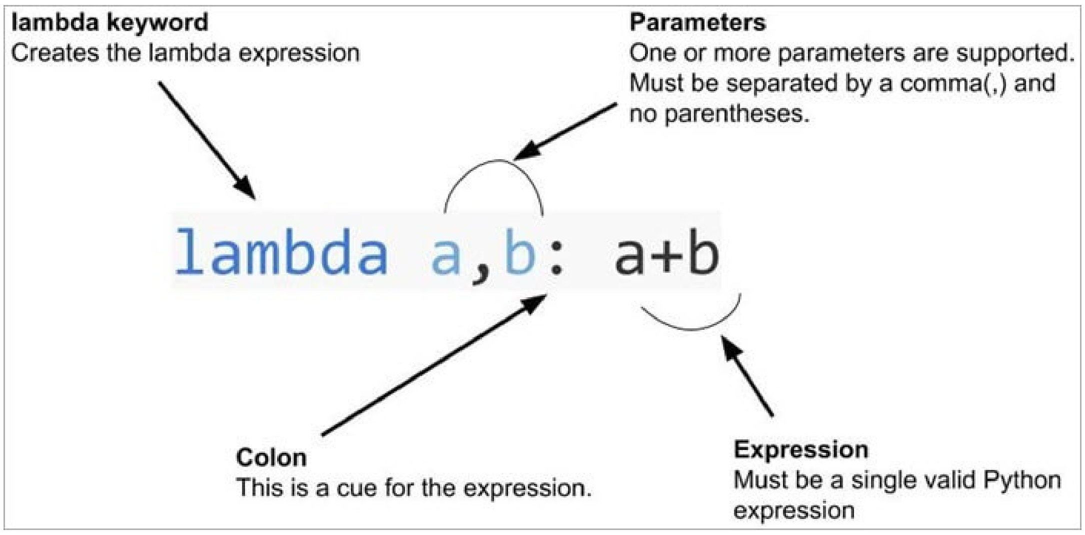
- It is till that fuction execute as well as variable of that function

NOTE: if there is not return in function, what will happen?

- we can use our function with print but, python will anyway return value and that will be "none".

Lambda Function

Lambda function is small anonymous function:


lambda keyword
Creates the lambda expression
Parameters
One or more parameters are supported.
Must be separated by a comma(,) and
no parentheses.
lambda a,b: a+b
Colon
This is a cue for the expression.
Expression
Must be a single valid Python
expression

```
In [1]: #lets see few simple example:
        a=lambda x,y:x+y
        a(3,4)

Out[1]: 7
```

```
In [2]: #did we get? we just give variable a and inside variable we stored lambda function
        # and later we pass 2 arguments to variable a
```

```
In [4]: #lets do few more
        a=lambda x:"even" if x%2==0 else "odd"
```

```
In [5]: a(11)

Out[5]: 'odd'
```

```
In [6]: #Now lets see few example of normal programming, normal function and lambda with some higher order function
```

What is higher order function?

- Its a function which contain function

```
In [7]: num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]
        multipleof7=[]
        multipleof5=[]
        othernum=[]

        for i in num:
            if i%7==0:
                multipleof7.append(i)
            elif i%5==0:
                multipleof5.append(i)
            else:
                othernum.append(i)


        print("Multiple of 7 is: ", multipleof7)
        print("Multiple of 5 is: ", multipleof5)
        print("Multiple of other number is: ", othernum)

        Multiple of 7 is:  [49, 7, 14, 28, 70]
        Multiple of 5 is:  [5, 15, 25, 30, 45, 50]
        Multiple of other number is:  [1, 3, 36]
```

```
In [8]: #Above we use normal number
```

Can we do similar above with function: normal function using def?

```
In [9]: #lets try

        def numsep(*a):
            multipleof7=[]
            multipleof5=[]
            othernum=[]

            for i in num:
                if i%7==0:
                    multipleof7.append(i)
                elif i%5==0:
                    multipleof5.append(i)
                else:
```

```
            othernum.append(i)


    print("Multiple of 7 is: ", multipleof7)
    print("Multiple of 5 is: ", multipleof5)
    print("Multiple of other number is: ", othernum)
```

In [10]:
```
numsep(1,3,5,49,15,25,30,7,14,28,36,45,50,70)
```

```
Multiple of 7 is:  [49, 7, 14, 28, 70]
Multiple of 5 is:  [5, 15, 25, 30, 45, 50]
Multiple of other number is:  [1, 3, 36]
```

In [13]:
```
#yes it worked, can we use return instead of print? yes we can

#lets try

def numsep(*a):
    multipleof7=[]
    multipleof5=[]
    othernum=[]

    for i in num:
        if i%7==0:
            multipleof7.append(i)
        elif i%5==0:
            multipleof5.append(i)
        else:
            othernum.append(i)


    return multipleof7, multipleof5, othernum
```

In [14]:
```
numsep(1,3,5,49,15,25,30,7,14,28,36,45,50,70)
```

Out[14]:
```
([49, 7, 14, 28, 70], [5, 15, 25, 30, 45, 50], [1, 3, 36])
```

In [15]:
```
#Now Can we use above method with list comprehension only?
```

In [16]:
```
#lets try same operation with list comprehension
```

In [17]:
```
num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]

multipleof7=[i for i in num if i%7==0]
multipleof5=[i for i in num if i%5==0]
othernum=[i for i in num if i%7!=0 and i%5!=0]
```

In [18]:
```
print(multipleof7)
```

```
[49, 7, 14, 28, 70]
```

In [19]:
```
print(othernum)
```

```
[1, 3, 36]
```

In [20]:
```
#yes list comprehension works too, can you see, how shortcut is method being, now lets use higer order function
```

Higher order function for same operation

In [21]:
```
#lets use filter, there are 3 higher order function we will use today: filter, map and reduce
```

Filter

In [24]:
```
num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]

multipleof7=list(filter(lambda x:x%7==0,num))
```

In [25]:
```
print(multipleof7)
```

```
[49, 7, 14, 28, 70]
```

In [26]:
```
#Why filter is know as higher order function-because, filter function is using lambda function
```

In [27]:
```
#Same lambda only can be used?

num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]
a=lambda x:x%7==0,num
```

In [28]:
```
print(a)
```

```
(<function <lambda> at 0x000002A14F473400>, [1, 3, 5, 49, 15, 25, 30, 7, 14, 28, 36, 45, 50, 70])
```

In [30]:
```
#but we can convert it in list
```

```
In [30]: #but we can convert it in list
num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]
list(lambda x:x%7==0,num)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[30], line 3
      1 #but we can convert it in list
      2 num=[1,3,5,49,15,25,30,7,14,28,36,45,50,70]
----> 3 list(lambda x:x%7==0,num)

TypeError: list expected at most 1 argument, got 2
```

```
In [31]: #Sorry we cannot do only with lambda, we need filter
```

Map (Higher order function)

```
In [32]: #for map function, we will need lamda and iterable
#example
# square the items of a list
list(map(lambda x:x**2,[1,2,3,4,5]))
```

```
Out[32]: [1, 4, 9, 16, 25]
```

```
In [33]: L = [1,2,3,4,5]
list(map(lambda x:'even' if x%2 == 0 else 'odd',L))
```

```
Out[33]: ['odd', 'even', 'odd', 'even', 'odd']
```

```
In [ ]: #There is another higher order function too, that is reduce, we will see it in future if it needed, otherwise,
#one more tutorial in Function, list comprehension, map and filter with lambda.
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js