პითონი 6 - ფუნქციები და მოდულები

Import math - მათემატიკური მოდულის იმპორტირება

Math.ceil (2.2) - ამრგავლებს მეტობით

Math.floor (2.2) - ამრგავლებს ნაკლებობით

ინტეჯერი ყოველთვის ამრგვალებს ნაკლებობით მაგალითად 2.2- დაამრგვალებს 2 -ად

იმისათვის რომ გამოითვალოს კონსინუსი, სინუსი და ტანგესი ჯერ უნდა შემოვიყვანოთ რადიანსის ცვლადი და მერე გადაეცეს სათითაოდ

მაგალითად:

Rad=math.radians(60)

Print(math.sin(rad))

Print(math.cos(rad))

Print(math.tan(rad))

Print(math.degrees(rad)) - გადაყავს გრადუსში

Math.fabs () - ნებისმიერი რიცხვი გადაყავს დადებით რიცხვში და float- ში

Math.gcd - უდიდესი საერთო გამყოფი, great common devider

Math.lcm- უმცირესი საერთო ჯერადი

Print (math.lcm (5,10)) – prints 10 რადგან უმცირესი ჯერადი რომელიც 10 ზეც იყოფა არის 10

Math.factorial(3) - ფაქტორიალს აგდაეცემა რიცხვი , ერთიდან რაღაც რიცხვამდე ყველა რიცხვის ნამრავლი , 1 ის 2ორის და 3 -ს ნამრავლი გამოდის 6

List - ში მოთავსებული რიცხვების ჯამის გამოთვლა

List =[1,2,5,0,-10, 7.1, 10, -5]

Math.fsum(list) - გამოთვლის ჯამს , და აუცილებლად უნდა გადაეცეს ცვლადის დასახელება რომელზეც მინიჭებულია ეს რიცხვების მასივი, ვიღებთ ფლოატის შედეგს

Math.sqrt()- კვადრატული ფესვი

Math.pow(2,3)- ახარისხება , ჯერ ვწერთ რა აგვყავს და რომელ რიცხვში აგვყავს

ასევე შეიძლება \*\*

Pi - მუდმივა

მაგალითად pi=3.14

Print (math.pi)

Functions-

კოდის ის ნაწილი რომელიც შეიძლება განმეორებითად გამოვიყენოთ, ატრიბუტები გადაეცემა

ფუნქციის შესაქმნელად არის ქივორდი def

Def-ის მერე იწერება ფუნქციის სახელი, რომელიც იწერება პატარა ასოთი

მაგალითად;

Def plus():

Return 10+5

Def minus ():

Return 10-5

Func1= plus()

Func2= minus()

Print (func1+func2)

ატრიბუტებით მინიჭება

Def plus (num1, num2)

Return num1+num2

Result1= plus (10-5) ამ შემთხვევაში 10 იანი მიენიჭება ნამ1-ს და 5 იანი მიენიჭება ნამ 2-ს

შეგვიძლია გადავცეთ ქი ვორდ არგუმენტებიც

მაგალითად; result2=plus (num2=17, num1=20) აქ შეცვლილია თანმიმდევრობა მაგრამ მაინც ზუსტად გადაეცემა რიცხვი კონკრეტულ ატრიბუტს

პოზიციური არგუმენტის და ქივორდის აღწერის არგუმენტს თუ ერთად ვწერთ მაშინ უნდა დავიცვათ თანმიმდევრობა , ჯერ იწერება პოზიციური და მერე აღწერითი

result2=plus (num1, num2=17,)

**Python 7 ფუნქციებთან მუშაობის დამატებითი მეთოდები**

ფუნქციის დაწერის დროს არგუმენტად შეიძლება გადაეცეს პოზიციური არგუმენტები.რომლებსაც ცვლადებში ვინახავთ, არგუმენტის რაოდენობა რომ იყოს შეუზღუდავი უნდა იყოს მითითებული პირველი არგუმენტის წინ ფიფქი

Def test (\*args) არ შეიძლება პოზიციურ არგუმნეტში გადაეცეს ქივორდის არგუმენტი

ამისათვის შეიძლება დაემატოს Def test (\*args, \*\*kwargs )

აქაც უნდა დავიცვათ თანმიმდევრობა ჯერ პოზიციური არგუმენტ და მერე ქივორდი

Def plus (num1, num2, num3, \*args):

Return num1+num2+num3

List=[10,15,20,25] args რომ არ დავამატო მაშინ რიცხვების რაოდენობა უნდა ემთხვეოდეს არგუმენტების რაოდენობას თუ დავამატებ მაშინ შეიძლება დაემატოს კიდევ

Sum\_num=plus (\*list)- დააჯამებს ლისტში მითითებულ რიცხვებს , ფიფქის დამატებით

ფუნქციის გარეთ შექმნილ ცვლადს ეწოდება გლობალური ცვლადი

X=10

Def legb ():

X=5

X = x+10

Print (x) დაბეჭდავს 15 ს , ასეთ ცვლადს ეწოდება enclosed ცვლადი

Def test2 ():

X=20

Print (x)

test2 ()

იქსი არის ტესტორისთვის ლოკალური ცვლადი . იქსის კითხვას იწყებს ბოლო ჩაშენებული ფუნქციიდან ჯერ ლოკალურს შეამოწმებს თუ არსებობს, შემდეგ ინქლოუზდში ჩაწერილ იქს ნახავს და იქაც თუ არა ცვლადი ჩაწერილი გადავა გლობალურ ცვლად იქსზე

თუ მინდა ჩაწერა გლობალურის ვწერთ ესე

Def test2 ():

global x

x+=10

Print (x)

ფუნქციის გამოძახების დროს ვწერთ ესე test2() ან ფრჩხილების გარეშე მაგრამ return-ით

Return test2

გავითო წრის სიფართოვე

Def circle\_area (pi):

Def whole\_area (radius):

Return pi\*radius \*\*2

Return whole\_area

area=circle\_area (math.pi)

Whole=area(10)

Print(whole)

აქ ამ ფუნქციონალში ჩაშენებულია ორი ფუნქცია , სერქლ არეა მოცავს სივრცის დაანგარიშებას და პირველი აბრუნებს შემდეგ საერთო სივრცეს, ხოლო მასში ჩაშენებული ფუნქციონალი გამოთვლას აკეთებს

ცვლადს არეას გადაეცა სრული ფუნქცია, შემდეგ მეორე ცვლადს მიენიჭა მეორე ცვლადის მნიშვნელობა რომელსაც ქონდა ატრიბუტი 10 , მან მასში ჩაშენებული ფუნქციით დაითვალა 10 ის გათვალისწინებით და ისე დაბეჭდა

Del circle\_area - შლის ნებისმიერ ობიექტს ოპერატიულ მეხსიერებაში

recursive function/Recursion function რეკურსიულია ფუნქცია თავის თავს იძახებს

Def rec\_func (x):

If x ==0:

Return “dasrulda”

Return rec\_func(x-10)

Func=rec\_func(10)

Print (func)

აბრუნებს რეკ ფუნქციას მანამ სანამ იქსი არ უდრის ნოლს და გამოიტანს დასრულდას.

აუცილებელია რითერნის დაწერა ორივეჯერ რომ დააბრუნოს ფუნქცია , თუ არა და ნანი იქნება

სტრუქტურაზეა დამოკიდებული ვაილს გამოვიყენებთ თუ რეკურსიულს, ძირითადად რეკურსიას იყენებენ

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720.

def factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* factorial(x-1))

num = 3

print("The factorial of", num, "is", factorial(num))

The factorial of 3 is 6

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

factorial(3) # 1st call with 3

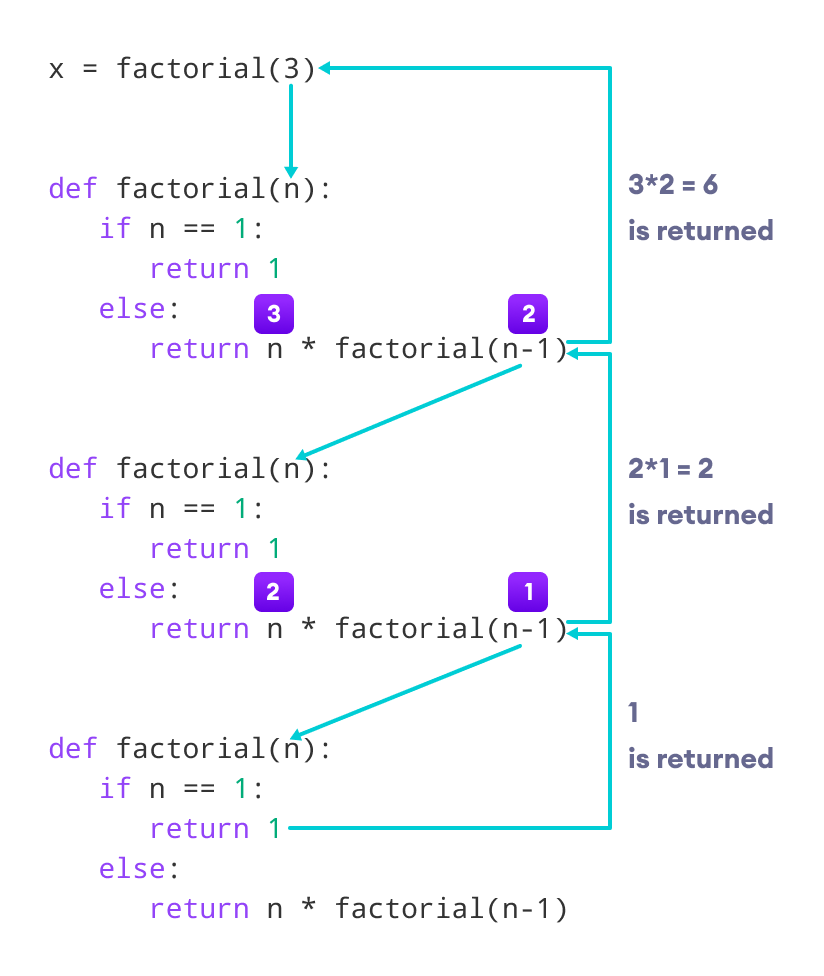
3 \* factorial(2) # 2nd call with 2

3 \* 2 \* factorial(1) # 3rd call with 1

3 \* 2 \* 1 # return from 3rd call as number=1

3 \* 2 # return from 2nd call

6 # return from 1st call



**Advantages of Recursion**

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

**Disadvantages of Recursion**

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

https://www.programiz.com/python-programming/recursion