**ALGORITMA DAN PEMROGRAMAN**

**FUNGSI DAN REKURSI**



Mata Kuliah : Algoritma dan pemrograman  
 Dosen : Prayitno, S.ST.,M.T,Ph.D

Disusun oleh

Nama : Maulana Azka Rifki Saputra

NIM : 3.34.24.1.13

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNIK ELEKTRO**

**POLITEKNIK NEGERI SEMARANG**

**2024**

Program 4.1 : Binomial

n=int(input("n="))

k=int(input("k="))

n\_fact=1

for i in range(1,n+1):

n\_fact\*=i

k\_fact=1

for i in range(1,k+1):

k\_fact\*=i

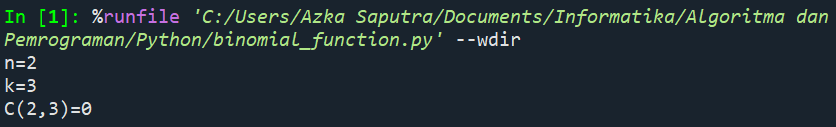
nk\_fact=1

for i in range(1,n-k+1):

nk\_fact\*=i

cnk=n\_fact//(k\_fact\*nk\_fact)

print(f"C({n},{k})={cnk}")

Output : 

Fungsi adalah blok instruksi yang dieksekusi sesuai kebutuhan ketika dipanggil. Data dapat dikirimkan ke sebuah fungsi yang dapat mengembalikan sejumlah data. Mekanisme mendorong dan mendapatkan data ke/dari fungsi ini dikenal sebagai pengoperan parameter dan pengembalian nilai.

Kita dapat dengan mudah membayangkan bahwa hal ini sangat tidak efisien dalam hal membaca dan menulis, dan yang terburuk, jika pendefinisian pertama salah, kita akan menyalin berkali-kali pendefinisian yang salah tersebut. Jika kita ingin memperbaiki masalah tersebut, kita harus mengingat dengan tepat di mana salinannya berada. Dalam contoh kita, kita dapat mengamati bahwa rentang yang digunakan dalam perulangan for dapat dimulai dari 2 misalnya, maka kita perlu melakukan koreksi di tiga tempat yang berbeda.

Kita hanya perlu mendefinisikan fungsi baru yang menghitung faktorial dari nilai tertentu dan menggunakannya saat dibutuhkan:

def fact(v):

f=1

for i in range(1,v+1):

f \*=i

return f

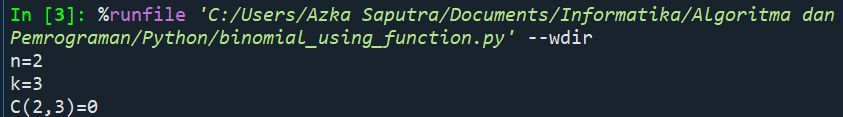
n=int(input("n="))

k=int(input("k="))

cnk=fact(n)//( fact(k) \* fact(n-k) )

print(f"C({n},{k})={cnk}")

Output :



4.1.1 Parameter

Perhatikan bahwa parameter v pada Program 4.19 adalah sebuah variabel terikat (lokal). Artinya:

Terikat karena namanya tidak menarik di luar fungsi, ini adalah sebuah pengenal yang digunakan untuk menangkap nilai dari ekspresi yang diberikan pada titik pemanggilan. Terikat seperti pada matematika dimana x → x 2 adalah fungsi yang sama persis dengan y → y2 .

Lokal karena variabel ini memiliki masa hidup pemanggilan dan tidak memiliki eksistensi di luarnya.

Variabel ini dibuat saat kontrol memasuki fungsi dan dihancurkan saat dikembalikan.

Sebuah fungsi dapat memiliki sejumlah parameter, bahkan tanpa parameter sama sekali:

def add(a, b):

return a+b

def hello():

print('Hello')

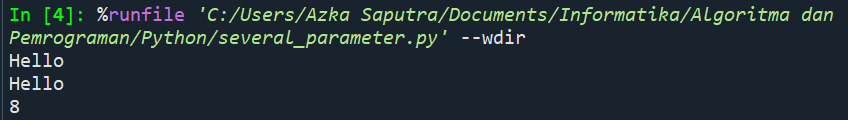
hello()

c=add(3,5)

hello()

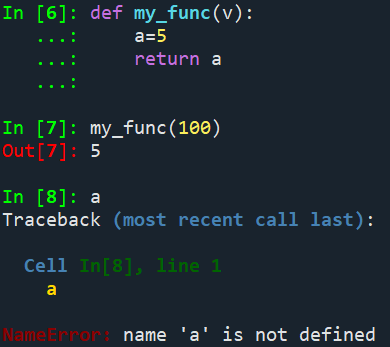
print(c)

Output :



4.1.2 Variabel Lokal

Variabel f yang muncul pada blok fungsi fakta pada Program 4.19 juga merupakan variabel terikat (lokal), seperti halnya parameter v. Itu berarti variabel ini tidak ada di luar blok ini. Tujuannya adalah untuk menangkap hasil antara di dalam fungsi; ini adalah mekanisme dalam yang tidak boleh diperlihatkan di luar. Variabel ini dibuat pada saat pertama kali digunakan, dan menghilang ketika pemanggilan fungsi keluar:



Variabel a dikonfigurasikan di dalam fungsi. Ini disebut masa pakai variabel.

4.1.3 Mengembalikan Nilai

Kita terkadang membedakan fungsi dan prosedur. Sebuah fungsi menghitung sebuah nilai, sedangkan prosedur melakukan sesuatu sebagai efek samping. Sebagai contoh:

math.sin adalah sebuah fungsi yang menghitung dan kemudian mengembalikan sinus dari sudut tertentu,

print adalah sebuah prosedur, tidak mengembalikan nilai, memiliki efek samping (menampilkan sesuatu pada output).

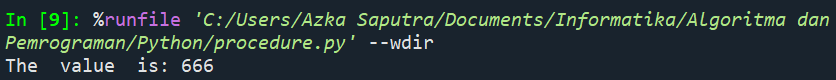
Ketika kita ingin kembali dari sebuah prosedur, kita dapat menggunakan pernyataan return yang sederhana:

def my\_print(v):

print("The value is:",v)

return

my\_print(666)

Output : 

Untuk sebuah fungsi, kita dapat menggunakan pernyataan nilai balik:

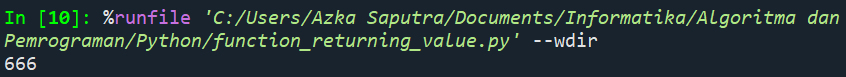
def my\_add(a, b, c):

return a+b+c

v=my\_add(6,60,600)

print(v)

Output :



4.1.4 Error handling

Menangani kesalahan adalah tugas yang sulit, terutama karena ada banyak jenis kesalahan. Kami tidak ingin menggali terlalu dalam mengenai masalah tersebut, tetapi setidaknya memberikan beberapa solusi dasar.

Contohnya adalah mencoba mengimplementasikan fungsi 1 . Seperti yang kita ketahui, fungsi ini tidak terdefinisi untuk x = 0 sehingga kita harus membuat sesuatu jika pengguna fungsi memanggilnya dengan 0. Tentu saja kita bisa menghentikan eksekusi secara tiba-tiba dengan memanggil quit:

def inverse(x):

if x==0:

print("Wrong parameter")

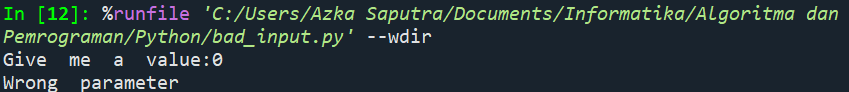
quit()

return 1/x

v = int(input("Give me a value:"))

print(inverse(v))

Output :



Pilihan yang lebih baik adalah mengembalikan nilai None dan membiarkan pemanggil menangani masalah tersebut:

def inverse(x):

if x==0:

return None

return 1/x

v = int(input("Give me a value: "))

inverse\_v = inverse(v)

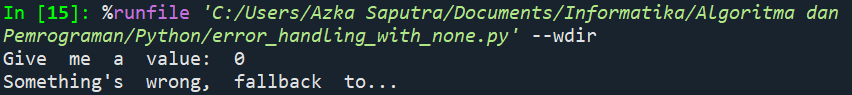
if inverse\_v==None:

print("Something's wrong, fallback to...")

else:

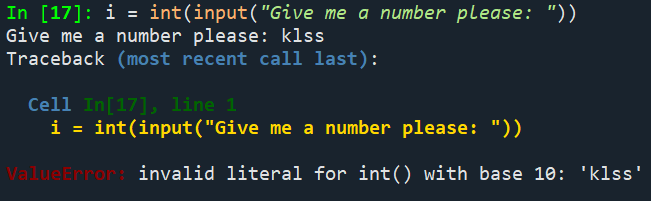
print(inverse\_v

Output :



4.1.5 Exception Handling

Penanganan pengecualian adalah jenis khusus dari penanganan kesalahan. Kita telah menemukan pengecualian, ingat apa yang terjadi ketika kita mencoba mengkonversi string yang tidak merepresentasikan nilai int ke nilai int seperti berikut ini:



Mekanisme tersebut sedikit rumit dan menguasainya berada di luar cakupan buku ini. Kami hanya ingin memberikan solusi dasar untuk mengatasi pengecualian.

Pengecualian adalah semacam alarm. Ketika ada sesuatu yang sangat buruk, sehingga komputasi normal tidak dapat dilanjutkan, sebuah alarm (pengecualian) akan dibunyikan (dilemparkan) dan aliran eksekusi normal akan terputus. Pada titik ini, PYT HO N mencari blok kode yang menyatakan untuk dapat menerima (menangkap) alarm tersebut. Jika tidak ada blok penangkap seperti itu, eksekusi diakhiri dengan sebuah pesan.

try berarti: mencoba mengeksekusi blok bersarang. Jika ada pengecualian yang dilemparkan ke dalam blok tersebut, eksekusi blok tersebut akan segera diinterupsi dan blok except dicari untuk dieksekusi. Jika blok try dieksekusi tanpa pengecualian sampai akhir, blok except diabaikan.

Pada contoh kecil sebelumnya, kita dapat menetapkan nilai default jika pengguna memasukkan sesuatu yang salah:

try:

i=int(input("Please, give me a number: "))

except:

i=0

print(i)

4.1.6 Closure

Sebuah pengenal yang tidak digunakan sebagai parameter atau sebagai pengenal lokal dapat direferensikan dalam sebuah fungsi, tetapi perlu diikat ke beberapa variabel yang ada “di luar” fungsi tersebut. Penutupan atau penutupan leksikal adalah pengikatan variabel bebas yang digunakan dalam sebuah fungsi ke lingkungannya.

4.1.6.1 Membaca Variabel “Di Luar”

Sebuah fungsi dapat menggunakan variabel yang didefinisikan di luar fungsi tersebut, asalkan kita hanya mengaksesnya untuk dibaca:

my\_var=666

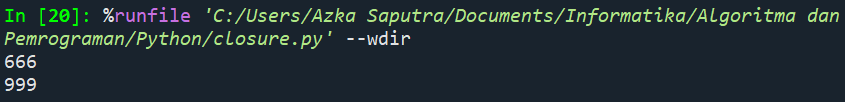
def my\_func(v):

return my\_var+v

print(my\_func(0))

print(my\_func(333))

Output :



Fungsi my\_func terikat pada variabel my\_var yang didefinisikan di luar fungsi tersebut. Kita bersikeras bahwa dalam kasus ini, hanya pembacaan variabel yang diperbolehkan (untuk memodifikasi variabel, lihat bagian selanjutnya). Kita katakan bahwa my\_var berada di dalam penutupan my\_func.

Meskipun diizinkan untuk melakukannya, disarankan untuk menghindari sebanyak mungkin pendefinisian fungsi-fungsi seperti itu:

* Hal ini akan menghasilkan fungsi-fungsi yang berperilaku berbeda tergantung pada titik pemanggilannya, menghasilkan program yang sangat sulit untuk dibaca dan dimengerti. Fungsi-fungsi seperti ini jauh dari fungsi matematika yang selalu memetakan nilai yang sama dengan entri yang sama.

my\_var=666

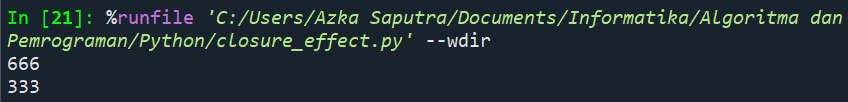
def my\_func(v):

return my\_var+v

print(my\_func(0))

my\_var=333

print(my\_func(0))

Output : 

Panggilan yang sama, nilai yang berbeda

Cobalah untuk tidak pernah mendefinisikan fungsi yang bergantung pada konteks (kecuali jika Anda tahu mengapa Anda membutuhkannya), atau Anda akan mengalami masalah serius pada suatu saat. Lebih memilih pendefinisian fungsi murni (fungsi yang tidak bergantung pada sesuatu yang “di luar”).

* ini mengarah pada fungsi yang bergantung pada keberadaan beberapa hal eksternal untuk dapat berjalan dengan benar.

def my\_func(v):

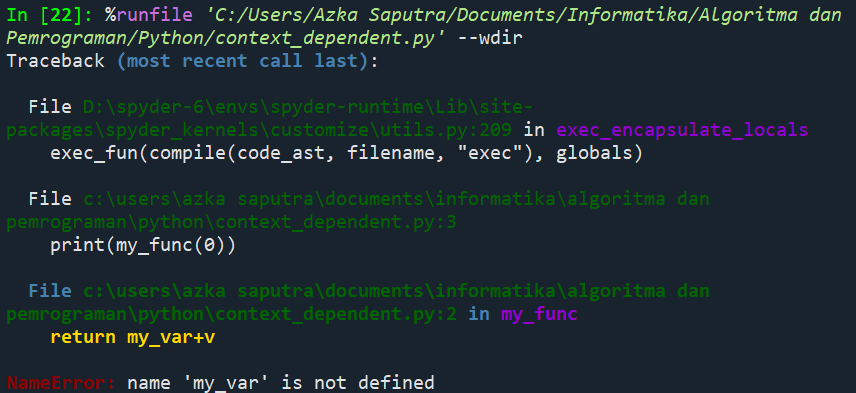
return my\_var+v

print(my\_func(0))

my\_var=333

print(my\_func(0))

Output :



4.1.6.2 Memodifikasi Variabel “Di Luar”

Kita telah melihat bahwa membaca variabel yang didefinisikan di luar fungsi dapat dilakukan tanpa usaha. Jika kita ingin memodifikasi sebuah variabel di dalam closure maka kita harus mendefiniskannya:

my\_var=10

def f():

global my\_var

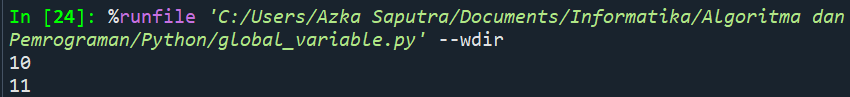
my\_var += 1

print(my\_var)

f()

print(my\_var)

Output :



Jika kita tidak mendefinisikan global, maka PY THO N akan mengasumsikan bahwa ia seharusnya merupakan variabel lokal, tetapi karena variabel ini belum didefinisikan secara lokal, maka ia akan menkomplain:

my\_var=10

def f():

global my\_var

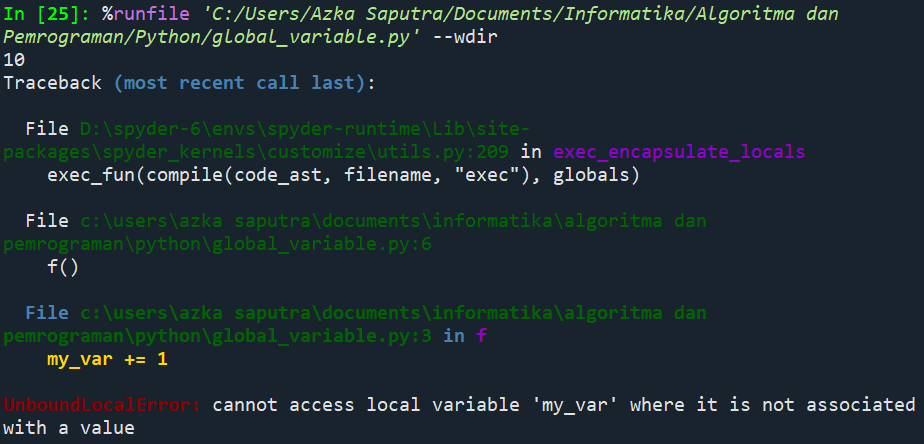
my\_var += 1

print(my\_var)

f()

print(my\_var)

Output :



4.1.7 Penyembunyian Variabel

Jika sebuah variabel lokal menggunakan pengenal yang sama dengan variabel yang didefinisikan di dalam closure, maka variabel tersebut akan disembunyikan. Jika kita tidak menggunakan variabel di dalam closure maka hal ini tidak menjadi masalah:

n=10

print(n)

def a\_function():

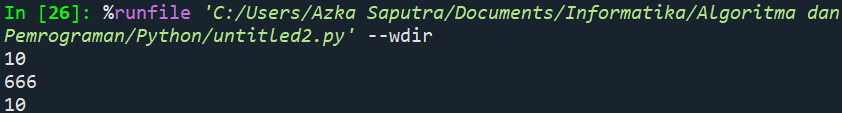
n=666

print(n)

a\_function()

print(n)

Output :



4.2 Modul

Fungsi membantu penulisan program dengan menguraikan definisi sebagai bagian kode yang dapat digunakan kembali dan dapat diparameterkan. Kadang-kadang kita ingin menggunakan kembali fungsi di berbagai program. Solusinya adalah dengan membuat modul-modul yang berisi fungsi-fungsi. Sebuah modul, dalam bentuknya yang sederhana, hanyalah sebuah source ﬁle yang hanya berisi fungsi-fungsi (yang bisa jadi jauh lebih kompleks, tetapi kita tidak akan membahasnya dalam buku ini).

Pertama-tama kita mendefinisikan modul yang berisi fungsi kita:

import math

def e\_approx(n):

e=0

for i in range(0,n+1):

e+=1/math.factorial(i)

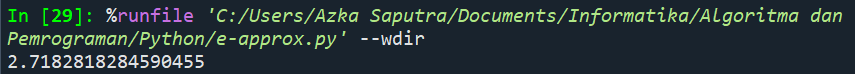
return e

dan menggunakannya di file sumber lain:

import approx

print(approx.e\_approx(50))

Output :



4.3 Lebih lanjut tentang Parameter

4.3.1 Nilai Default

Misalkan kita perlu menghitung kosinus sudut yang dinyatakan dalam radian, derajat, atau gradian. Solusi pertama dapat dilakukan dengan mendefinisikan tiga fungsi yang berbeda:

import math

def cosine\_radian(a):

"eats angles in radians"

return math.cos(a)

def cosine\_degree(a):

"eats angles in degrees"

return math.cos(a/180\*math.pi)

def cosine\_gradian(a):

"eats angles in gradians"

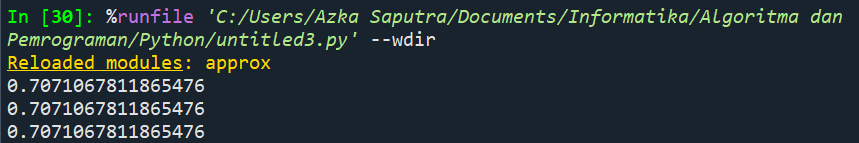
return math.cos(a/200\*math.pi)

print(cosine\_radian(math.pi/4))

print(cosine\_degree(45))

print(cosine\_gradian(50))

Output :



Cara lain adalah dengan memfaktorkan ini menjadi satu fungsi dengan memasukkan parameter lain yang mewakili unit:

import math

def cosine(angle,unit):

"""unit: 1: radians, 2: degrees, 3: gradians"""

if unit == 1:

return math.cos(angle)

if unit == 2:

return math.cos(angle/180\*math.pi)

if unit == 3:

return math.cos(angle/200\*math.pi)

print("Bad unit for angle")

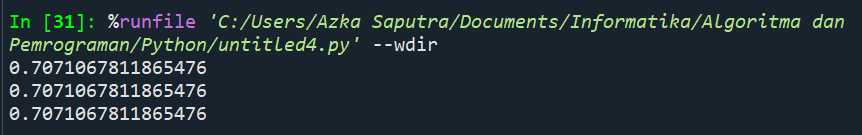
quit()

print(cosine(math.pi/4,1))

print(cosine(45,2))

print(cosine(50,3))

Output :



4.3.2 Argumen Posisi dan Kata Kunci

Kita telah menggunakan variabel untuk menangkap nilai yang dilewatkan pada pemanggilan fungsi. Pemanggilan yang telah kita lakukan sebelumnya menggunakan argumen posisi, yang berarti bahwa nilai yang dilewatkan dipetakan secara berurutan ke daftar parameter (nilai pertama ke parameter pertama, kedua ke kedua, dan seterusnya). Kita dapat menggunakan identifier untuk menamai parameter pada titik pemanggilan, dalam hal ini urutan pada titik pemanggilan tidak lagi menjadi keharusan:

import math

def cosine(angle,unit):

if unit == 1:

return math.cos(angle)

if unit == 2:

return math.cos(angle/180\*math.pi)

if unit == 3:

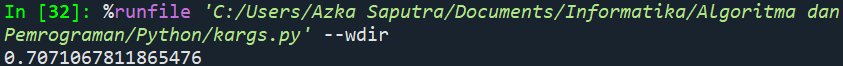
return math.cos(angle/200\*math.pi)

print("error")

quit()

print(cosine(unit=3,angle=50))

Output :



4.4 Rekursi

Rekursi adalah alat yang paling kuat dalam pemrograman, seperti yang telah kita sebutkan di bagian pendahuluan bab ini, itulah yang membuat kekuatan komputasi. Bukan berarti tanpa rekursi program menjadi remeh; ini hanya berarti bahwa dari sudut pandang ilmu komputer teoritis kita dapat menghitung sekumpulan fungsi yang lebih besar jika kita menggunakan rekursi.

Definisi tersebut dengan mudah diterjemahkan ke dalam kode PYTHON berikut ini:

def fact(n):

f = 1

for i in range(1,n+1):

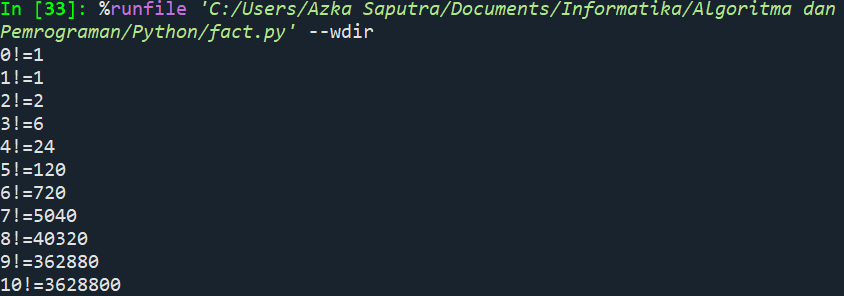
f \*= i

return f

for i in range(0,11):

print(f"{i}!={fact(i)}")

Output :



Itu adalah deﬁnisi rekursif, sebuah deﬁnisi yang menggunakan dirinya sendiri. Kita juga dapat dengan mudah menerjemahkannya dalam PYTHON seperti ini:

def fact(n):

if n == 0:

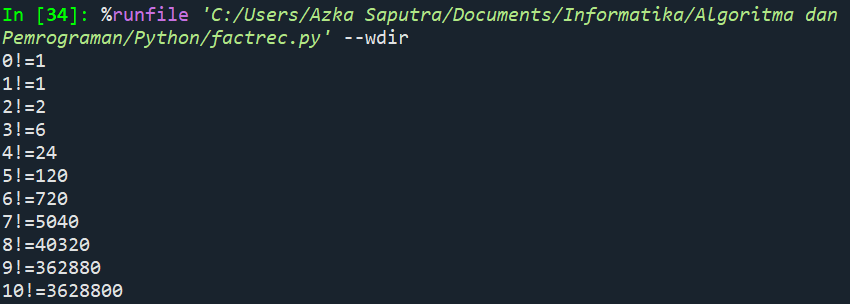
return 1

return n \* fact(n-1)

for i in range(0,11):

print(f"{i}!={fact(i)}")

Output :



Tidak ada batasan dalam memanggil fungsi. Kita dapat memanggil sebuah fungsi dimanapun fungsi tersebut didefinisikan, bahkan di dalam tubuh definisinya.Pemanggilan ke fungsi seperti itu dapat memanggil fungsi tersebut juga, dan itu adalah sempurna. Dalam beberapa hal, kita memiliki semacam ular yang menggigit ekornya sendiri. Sebuah lingkaran setan yang harus diputus pada suatu saat, jika tidak, kita akan mendapatkan komputasi yang tidak akan pernah selesai (sebuah hal yang harus kita hindari, setidaknya saat kita membicarakan algoritma-lihat Bagian II). Mari kita lihat apa yang terjadi:

def fact(n):

return n\*fact(n-1)

print(fact(10))

Output :

