Python - Escopo Variável

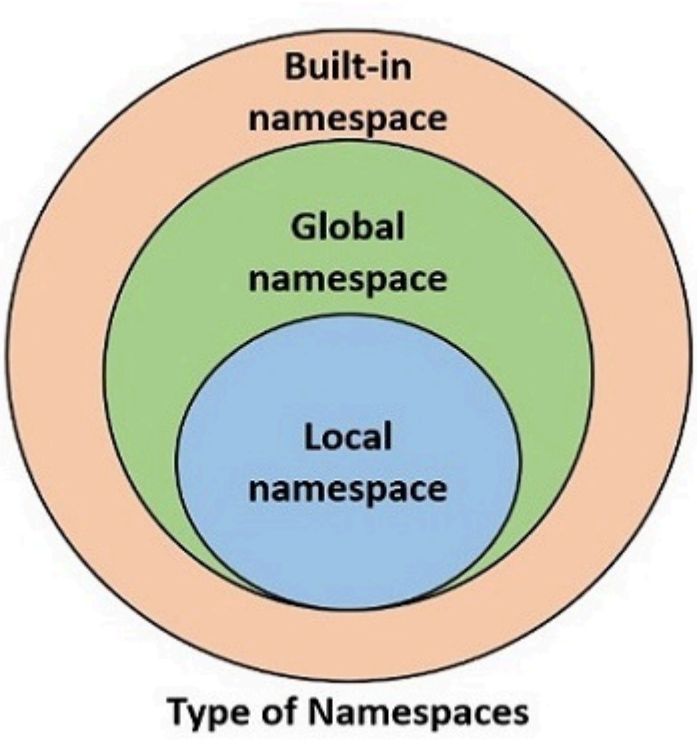
Escopo da variável Python

Uma variável em Python é um nome de símbolo para o objeto na memória do computador. Python trabalha no conceito de namespaces para definir o contexto para vários identificadores, como funções, variáveis, etc. Um namespace é uma coleção de nomes simbólicos definidos no contexto atual.

Python fornece os seguintes tipos de namespaces -

- **O namespace integrado** contém funções e exceções integradas. Eles são carregados na memória assim que o interpretador Python é carregado e permanecem até que o interpretador esteja em execução.
- **O namespace global** contém quaisquer nomes definidos no programa principal. Esses nomes permanecem na memória até que o programa seja executado.
- **O namespace local** contém nomes definidos dentro de uma função. Eles ficam disponíveis até que a função esteja em execução.

Esses namespaces estão aninhados um dentro do outro. O diagrama a seguir mostra o relacionamento entre namespaces.



Type of Namespaces

A vida de uma determinada variável é restrita ao namespace em que ela está definida. Como resultado, não é possível acessar uma variável presente no namespace interno a partir de

qualquer namespace externo.

Função Python globais()

A biblioteca padrão do Python inclui uma função integrada **globals()** . Ele retorna um dicionário de símbolos atualmente disponíveis no namespace global.

Execute a função globals() diretamente do prompt do Python.

```
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_in
```

Pode-se ver que o módulo interno que contém definições de todas as funções integradas e exceções integradas é carregado.

Salve o código a seguir que contém poucas variáveis e uma função com mais algumas variáveis dentro dele.

```
name = 'TutorialsPoint'
marks = 50
result = True
def myfunction():
   a = 10
   b = 20
   return a+b

print (globals())
```

Chamar globals() de dentro deste script retorna o seguinte objeto de dicionário -

```
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_frozen_importli
```

O namespace global agora contém variáveis no programa e seus valores e o objeto de função nele (e não as variáveis na função).

Qualquer variável criada fora de uma função pode ser acessada dentro de qualquer função e portanto possui escopo global. A seguir está um exemplo para mostrar o uso de variável global em Python:

```
x = 5
y = 10
def sum():
   sum = x + y
```

```
   return sum
print(sum())
```

Isso produzirá o seguinte resultado:

```
15
```

Função locais() do Python

A biblioteca padrão do Python inclui uma função integrada **locals()** . Ele retorna um dicionário de símbolos atualmente disponíveis no namespace local da função.

Modifique o script acima para imprimir o dicionário de namespaces globais e locais de dentro da função.

```
name = 'TutorialsPoint'
marks = 50
result = True
def myfunction():
   a = 10
   b = 20
   c = a+b
   print ("globals():", globals())
   print ("locals():", locals())
   return c
myfunction()
```

A **saída** mostra que locals() retorna um dicionário de variáveis e seus valores atualmente disponíveis na função.

```
globals(): {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <_froze
locals(): {'a': 10, 'b': 20, 'c': 30}
```

Como as funções globals() e locais retornam dicionário, você pode acessar o valor de uma variável do respectivo namespace com o método get() de dicionário ou operador de índice.

```
print (globals()['name']) # displays TutorialsPoint
print (locals().get('a')) # displays 10
```

A seguir está um exemplo simples para mostrar o uso de variáveis locais em Python:

```
def sum(x,y):
   sum = x + y
```

```
   return sum
print(sum(5, 10))
```

```
15
```

Conflito de namespace em Python

Se uma variável de mesmo nome estiver presente no escopo global e local, o interpretador Python dará prioridade àquela no namespace local.

```
marks = 50 # this is a global variable
def myfunction():
   marks = 70 # this is a local variable
   print (marks)

myfunction()
print (marks) # prints global value
```

Ele produzirá a seguinte **saída** -

```
70
50
```

Se você tentar manipular o valor de uma variável global de dentro de uma função, o Python gerará **UnboundLocalError** .

```
marks = 50 # this is a global variable
def myfunction():
   marks = marks + 20
   print (marks)

myfunction()
print (marks) # prints global value
```

Ele produzirá a seguinte **saída** -

```
  marks = marks + 20
      ^^^^^
UnboundLocalError: cannot access local variable 'marks' where it is not associated with a value
```

Para modificar uma variável global, você pode atualizá-la com uma sintaxe de dicionário ou usar a palavra-chave **global** para referenciá-la antes de modificá-la.

```
var1 = 50 # this is a global variable
var2 = 60 # this is a global variable
def myfunction():
   "Change values of global variables"
   globals()['var1'] = globals()['var1']+10
   global var2
   var2 = var2 + 20

myfunction()
print ("var1:",var1, "var2:",var2) #shows global variables with changed values
```

Ele produzirá a seguinte **saída** -

```
var1: 60 var2: 80
```

Por último, se você tentar acessar uma variável local no escopo global, o Python gerará NameError, pois a variável no escopo local não pode ser acessada fora dela.

```
var1 = 50 # this is a global variable
var2 = 60 # this is a global variable
def myfunction(x, y):
   total = x+y
   print ("Total is a local variable: ", total)

myfunction(var1, var2)
print (total) # This gives NameError
```

Ele produzirá a seguinte saída -

```
Total is a local variable: 110
Traceback (most recent call last):
  File "C:\Users\user\examples\main.py", line 9, in <module>
  print (total) # This gives NameError
        ^^^^^
NameError: name 'total' is not defined
```