function

September 5, 2020

1 Functions

[ ]:
```
def fibanocci(n):
    a,b=0,1
    while a<n:
        a,b=b,a+b
        print(a,end=" ")
```

[4]:
```
fibanocci(10)
```

```
1 1 2 3 5 8 13
```

[10]:
```
def fiboncciListe(n):
    out=[1,1]
    for i in range(n-2):
        out.append(out[i]+out[i+1])
    return out
```

[11]:
```
print(fiboncciListe(10))
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

[22]:
```
def fibonacciRecursive(n):
    if n==0 or n==1:
        return 1
    else:
        return fibonacciRecursive(n-1)+fibonacciRecursive(n-2)
```

[26]:
```
for n in range(10):
    print(fibonacciRecursive(n),end=" ")
```

```
1 1 2 3 5 8 13 21 34 55
```

1.1 Default arguments values

The most useful form is to specify a default value for one or more aguments. This creates function that can be called with fewer arguments than it is defined to allow.

[3]:
```
def adress(name,country='France',city='Le Croisic'):
    print("Monsieur ou Madame {} vit en ".format(name),country, "dans la ville␣
 ↪de ", city )
```

[4]:
```
adress("Cheikh BEYE")
```

```
Monsieur ou Madame Cheikh BEYE vit en  France dans la ville de  Le Croisic
```

[6]:
```
adress("Ndoye Alioune",city='Le Mans')
```

```
Monsieur ou Madame Ndoye Alioune vit en  France dans la ville de  Le Mans
```

The dafault value is evaluted only once. But when the default is a mutable object kind of *list* or *set* or *dictionnary* the defalut can be evaluated more than once.

[9]:
```
def add(elm,L=list()):
    L.append(elm)
    return L
```

[11]:
```
for i in range(4):
    print(add(i))
```

```
[1, 0]
[1, 0, 1]
[1, 0, 1, 2]
[1, 0, 1, 2, 3]
```

To handlle this we can proceed as follows

[13]:
```
def add_noIteract(elm,L=None):
    if L is None:
        L=list()
        L.append(elm)
    return L
```

[14]:
```
for i in range(4):
    print(add_noIteract(i))
```

```
[0]
[1]
[2]
[3]
```

1.2 Keyword Arguments

Functions can also be called using Keyword arguments of the form *kwarg=value*. For instance, the following function:

[2]:
```
def parrot(voltage,state='a stiff', action='voom',type='Norwegian'):
    print("__This a parrot wouldn't",action,end=' ')
    print("If you put",voltage,"volts through it.")
    print("__lovely plumage, the", type)
    print("__It's", state,"!")
```

1.2.1 1 positionnal argument

[6]:
```
parrot(1000)
```

```
__This a parrot wouldn't voom If you put 1000 volts through it.
__lovely plumage, the Norwegian
__It's a stiff !
```

1.2.2 1 keyword argument

[7]:
```
parrot(voltage=100)
```

```
__This a parrot wouldn't voom If you put 100 volts through it.
__lovely plumage, the Norwegian
__It's a stiff !
```

1.2.3 2 keyword arguments

[8]:
```
parrot(voltage=10000,action='VOOOOM')
```

```
__This a parrot wouldn't VOOOOM If you put 10000 volts through it.
__lovely plumage, the Norwegian
__It's a stiff !
```

1.2.4 2 keyword arguments

[9]:
```
parrot(action='VOOOOOM',voltage=100000)
```

```
__This a parrot wouldn't VOOOOOM If you put 100000 volts through it.
__lovely plumage, the Norwegian
__It's a stiff !
```

1.2.5 3 positionnal arguments

[10]:
```
parrot('a million', 'berefit of life','jump')
```

```
__This a parrot wouldn't jump If you put a million volts through it.
__lovely plumage, the Norwegian
__It's berefit of life !
```

1.2.6 1 positionnal argument and 1 keyword argument

[11]:
```
parrot('a thousand',state='pushing up the daisies')
```

```
__This a parrot wouldn't voom If you put a thousand volts through it.
__lovely plumage, the Norwegian
__It's pushing up the daisies !
```

In functionnal call, keyword arguments must follow positionnal arguments. All the keyword arguments passed must match one of the arguments accepted by the fucntion and the order is not important. This also includes non-optionnal arguments. No argument may receive a value more than once.

When a final formal parameter of the form _name**_ is present, it receives a dictionnary containing all keyword arguments except for those corresponding to a formal parameter. This may be combined with a formal parameter of the form _name*_ which receives a tuple containing the postionnal arguments beyond the fomal parameter list.

[6]:
```
def cheeseshop(kind, *arguments, **keyword):
    print("--Do you have any",kind, " ?")
    print("--I'm sorry, we are all out of",kind)
    for arg in arguments:
        print(arg)
    print("--"*40)
    for kw in keyword:
        print(kw, ":",keyword[kw])
```

[7]:
```
cheeseshop("Limburger","It's very runny, sir.",
           "It's really very VERY runny, sir.",
           shopkeeper="Micheal More",
           cleint="Arame",
           sketch="Cheese shop sketch"
           )
```

```
--Do you have any Limburger  ?
--I'm sorry, we are all out of Limburger
It's very runny, sir.
It's really very VERY runny, sir.
--------------------------------------------------------------------------------
shopkeeper : Micheal More
cleint : Arame
sketch : Cheese shop sketch
```

By default, arguments may be passed to a python function either by position or explicitly by keyword. For reability and performance, it makes sense to restrict the way arguments can be passed so that a developer need only look at the function definition to determine if items are passed by position or keyword. We use / and _*_ to indicate the kind of parameter by how the arguments may be passed to the function: position-only, positional-or-keyword, and keyword-only. keyword parameter are also refered to as named parameter.

[3]:
```
def position_only(arg, /):
    print('Hello' ,arg)
```

1.3 Call with a postion-only argument

[4]:
```
position_only('Cheikh')
```

```
Hello Cheikh
```

1.4 Call with a keyword agument

[7]:
```
position_only(arg='Cheikh')
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-7-c879fa889a2a> in <module>
----> 1 position_only(arg='Cheikh')


TypeError: position_only() got some positional-only arguments passed as␣
↪keyword arguments: 'arg'
```

As expected the call with a keyword argument is not allowed.

If positional-only, the parameters' oreder matters. Positional-only the parameters are placed before a / (forward-slash). The / is used to logically separate the positional-only from the rest of the parameters. If there is no / in the fucntion definition,there are no positional-only parameters.

[13]:
```
def keyword_only(*, arg):
    print('Hello', arg)
```

1.5 Call with a keyword agument

[16]:
```
keyword_only(arg='Cheikh')
```

```
Hello Cheikh
```

1.6 Call with positional-only argument

[17]:
```
keyword_only('Cheikh')
```

```
␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-17-a7fb002d5892> in <module>
   ----> 1 keyword_only('Cheikh')


       TypeError: keyword_only() takes 0 positional arguments but 1 was given
```

As expected the keyword argument is not allowed.