Python - Functional Programming

FUNCTIONS

Contents

- Keyword Arguments
- **kwarsg
- Recaper
- Application - Timer
- look out for Parameter defaults

Keyword Arguments

- Positional arguments can, optionally, be specified by using the parameter name whether or not the parameters have default values.
- Keyword arguments aka Named arguments.
- All arguments after the first named (keyword) argument, must be named too Default arguments may still be omitted.

```
def my_func(a, b=5, c=10)::
    #some code here
```

If call the function then python does this

Case#1 : my_func(10, c= 20) → a = 10, b = 5 , C = 20

Case#2: my_func(20, b = 10, 10) → **SyntaxError** positional argument follows keyword argument.

Case#3: my_func (c=10,b=12,a=17) → a = 17 , b = 12, c = 10

Mandatory Keyword Arguments

- How to make a mandate keyword arg?
- To do so, we create parameters after the positional parameters have been exhausted.

```
def func(a, b, *args, d):
    #code
```

In this case, *args effectively exhausts all positional arguments,
recap from the positional arguments session *args can acquire arbitrary no .of positional args.

Due this d must be passed as a keyword (named) argument.

func(1,2,3,4,5,6,d=10)→ a =1 ,b =2, args = (3,4,5,6) , d = 10

No Positional Args

- we can force no positional arguments at all:

```
def func(*,d):
    #code
```

In the above code * indicates the "end" of positional arguments.

func(11,12,d=22) → **TypeError**: func() takes 0 positional arguments but 2 positional arguments (and 1 keyword-only argument) were given.

So by this technique we can ignore Positional args

**kwargs

- ***args** is used to scoop up variable amount of remaining positional arguments.
- args is just parameter name , Performer is * which is type Tuple.
- ****kwargs** is used to scoop up a variable amount of remaining keyword argument.
- Same here kwargs is just parameter name , ** is the performer type Dictionary.
- **kwargs can be specified even if the positional arguments have not been exhausted.

Note: No parameters can come after **kwargs

**kwargs

```
def func(*, d, **kwargs):
    #code
```

func(d=1, a=2, b=3) → d = 1
kwargs = {'a': 2, 'b': 3}

func(d=1) → d = 1
kwargs = {}

Positional and Keyword

Positional arguments	Keyword-only arguments
specific - may have default values	after positional arguments have been exhausted
*args collects, and exhausts remaining positional arguments	specific - may have default values
* indicates the end of positional arguments (effectively exhausts)	**kwargs collects any remaining keyword arguments.

Typical Use Case: print() function

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file* and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

- *objects arbitrary number of positional arguments
- After that are keyword-only arguments they all have default values, so they are all optional.
- Often, keyword-only arguments are used to modify the default behavior of a function

send me your suggestions!

- Email: sundar.muthuraman.offical@gmail.com
- Contact me : +91 9962988838