POWERING POTENTIAL

Functions

Module 14

FUNCTIONS

Contents

Python functions

- **Functions are objects**
- **Defined with the def statement, followed by the argument list**
  - Just like conditionals, membership is by *indentation*

```
def make_list(val, times):
    res = str(val) * times
    return res
```

- **Arguments are named**
  - Defaults may be assigned
- **`return` statement is optional**
  - Any object type may be returned
  - Default is the empty object `None`
- **Variables are local if assigned**
  - Unless the keyword `global` is used

***"Function names should be lowercase..."* – PEP008**

Function parameters

Values required by the function

- Specified within the parentheses of the function declaration

```
def print_list(val, times):
   print(str(val) * times)
```

- Parameters are passed by assignment (copy)

```
print_list(5, 3)
print_list(0, 4)
```

- Since they are references, changes alter the callers variables

```
def change_list(inlist, val, times):
    inlist += str(val) * times

mylist=[]
change_list(mylist, 'h', 8)
print(mylist)
```

```
['h', 'h', 'h', 'h', 'h', 'h', 'h', 'h']
```

Assigning default values to parameters

- **Assign the default value when defining the function**
  - Need not pass the parameter value while calling it

```
def print_vat(gross, vatpc=17.5, message='Summary:'):
   net = gross/(1 + (vatpc/100))
   vat = gross - net
   print(message, 'Net: {0:5.2f} Vat: {1:5.2f}'.format(net, vat))


print_vat(9.55)
```

```
Summary: Net:  8.13 Vat:  1.42
```

- **Default one, then you must default those to the right**
  - Applies when defining a function
- **When calling a function, you can use keywords instead**

```
print_vat(9.55, message='Final sum:')
```

```
Final sum: Net:  8.13 Vat:  1.42
```

Passing parameters - review

```
def my_func(file, dir, user='root'):
    print('file: {:}, dir: {:}, to: {:} '.
          format(file, dir, user))
```

By position

```
my_func('one', 'two', 'three')
```

```
file: one, dir: two, to: three
```

By default

```
my_func('one', 'two')
```

```
file: one, dir: two, to: root
```

Or by name

```
my_func(file='one', user='three', dir='two')
```

```
file: one, dir: two, to: three
```

Enforcing named parameters

Use a bare * to force a user to supply named arguments

- No need for a dictionary

```
def print_vat(*, gross=0, vatpc=17.5, message='Summary:'):
   net = gross/(1 + (vatpc/100))
   vat = gross - net
   print(message, 'Net: {0:5.2f} Vat: {1:5.2f}'.format(net, vat))

print_vat(vatpc=15, gross=9.55)
print_vat()
```

```
Summary: Net:  8.30 Vat:  1.25
Summary: Net:  0.00 Vat:  0.00
```

py3

Attempting to pass positional parameters will fail

```
print_vat(15, 9.55)
```

```
TypeError: print_vat() takes exactly 0 positional arguments (2 given)
```

Unpacking and variadic functions

- **Functions usually have a fixed number of parameters**
- ***Unpacking* passes a sequence's elements as single arguments**

```
def my_func(a, b, c):
    print(a, b, c)

mytup = 23, 45, 67
my_func(*mytup)
```

```
23 45 67
```

- **Variadic functions have a variable number of parameters**
- They can be collected into a tuple with a `*` prefix

```
def my_func(dir, *files):
    print('dir:', dir, 'files:', files)

my_func('c:/stuff', 'f1.txt', 'f2.txt', 'f3.txt')
```

```
dir: 'c:/stuff', files: ('f1.txt', 'f2.txt', 'f3.txt')
```

Keyword parameters

- **Look just like the key-value pairs of a dictionary**
  - Because that is what they are
- **Prefix a parameter with ** to indicate a dictionary**
  - Since a dictionary is unordered, then so are the parameters
  - May only come at the end of a parameter list

```
def print_vat(**kwargs):
   print(kwargs)

print_vat(vatpc=15, gross=9.55, message='Summary')
```

```
{'gross': 9.55, 'message': 'Summary', 'vatpc': 15}
```

- **Use ** to unpack caller's parameters from a dictionary**

```
argsdict = dict(vatpc=15, gross=9.55, message='Summary')
print_vat(**argsdict)
```

Returning objects from a function

- **Use a `return` statement, followed by the object to be returned**
- *Any* Python object may be returned

Returning an object:

- Stops the execution of the function
- Passes the object back to the caller
- If return is not used, a reference to `None` is returned

```
def calc_vat(gross, vatpc=17.5):
    net = gross/(1 + (vatpc/100))
    vat = gross - net
    return [f'{net:05.2f}', f'{vat:05.2f}']

result = calc_vat(42.30)

print(calc_vat(9.55))
```

```
['08.13', '01.42']
```

Variables in functions

- **By default, variables used in a function are local**
- **Global variables are defined using `global`**
  - Are local to the current module, or *namespace*

```
result = 3

def scope_test1():
    result = 42

scope_test1()
print(result)

def scope_test2():
    global result
    result = 42

scope_test2()
print(result)
```

→ 3

→ 42

Lambda functions

- **Anonymous short-hand functions**
  - Cannot contain branches or loops
  - Can contain *conditional expressions*
  - Cannot have a `return` statement or assignments
  - Last result of the function is the returned value

```
compare=lambda a, b: -1 if a < b else (+1 if a > b else 0)

x = 42
y = 3

print("a>b", compare(x, y))
```

Parameters

```
a>b 1
```

- **Often used with the `map()` and `filter()` built-ins**
  - Applies an operation to each item in a list

```
new_list = list(map(lambda a: a+1, source_list))
```

Lambda as a sort key

- **Takes the element to be compared**
- **Returns the key in the correct format**
- Sort each country by the second field, population

```
countries = []
for line in open('country.txt'):
    countries.append(line.split(','))

countries.sort(key=lambda c: int(c[1]))

for line in countries:
    print(','.join(line), end='')
```

```
Antarctica,0,-,-,Antarctica,1961,-,-,-
Arctica,0,-,-,Arctic Region,-,-,-,-
Pitcairn Islands,46,Adamstown,?,Oceania,-,...
Christmas Island,396,The Settlement,?,Oceania,...
Johnston Atoll,396,-,-,Oceania,-,US Dollar,-,...
```

SUMMARY

- **A function is a defined object**
- Variables have local scope unless `global` is used
- Other functions can be nested within
- **Parameters are declared local variables**
- May be assigned defaults, from the right
- **arg* means unpack to a tuple
- ***arg* means unpack to a dictionary
- * forces the caller to use named parameters
- **Can return any object**
- Including lists and dictionaries
- **Short, inline, anonymous functions can be defined using lambda**