Functions

Python basics

Kunal Khurana

2024-02-16

Table of contents

Functions

- evit repetition
- scope and local function

```
def func():
    a = []
    for i in range(5):
        a.append(i)
```

```
a
```

```
NameError: name 'a' is not defined
```

```
# making the function global to fix it

#a = None

def bind_a_variable():
    global a
    a = []
    bind_a_variable()

print(a)
```

```
None
```

```
c = []
def func1():
    for i in range(5):
        c.append(i)
```

```
c
```

```
[0, 1, 2, 3, 4]
```

returning multiple values

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c

a, b, c = f()
```

```
f()
```

```
(5, 6, 7)
```

```
f(1, 2, 3)  # traceback because 0 positional argument should have been provided
```

```
TypeError: f() takes 0 positional arguments but 3 were given
```

```
def g():
    a = 5
    b = 3
    c = 5
    return {'a':a, 'b': b, 'c': c}
```

```
g()
```

```
{'a': 5, 'b': 3, 'c': 5}
```

```
states = ["   Alabama ", "Georgia!", "Georgia", "georgia", "FlOrIda",          "south   ca
```

```
### data cleaning with functions

import re

def clean_strings(strings):
    result = []
    for value in strings:
        value = value.strip()
        value = re.sub("[!#?]", "", value)
        value = value.title()
        result.append(value)
    return result
```

```
clean_strings(states)
```

```
['Alabama',
 'Georgia',
 'Georgia',
 'Georgia',
 'Florida',
 'South   Carolina',
 'West Virginia']
```

lambda functions

```
def short_function(x):
    return x * 2
```

```
equiv_anon = lambda x: x * 2
```

```
short_function(3)
```

```
6
```

```
equiv_anan = 2
```

```
def apply_to_list (some_list, f):
    return [f(x) for x in some_list]
```

```
ints = [4, 0, 1, 4, 6]
apply_to_list (ints, lambda x: x * 2)
```

```
[8, 0, 2, 8, 12]
```

```
strings = ['foo', 'card', 'bar', 'aaa', 'abab']

strings.sort(key = lambda x : len(set(x)))

strings
```

```
['aaa', 'foo', 'abab', 'bar', 'card']
```

Generators

```
some_dict = {'a': 1, 'b': 2, 'c': 3}

for key in some_dict:
    print(key)
```

```
a
b
c
```

```
dict_iterator = iter(some_dict)

dict_iterator
```

```
<dict_keyiterator at 0x193c70f6040>
```

```
list(dict_iterator)
```

```
['a', 'b', 'c']
```

```
tuple(dict_iterator)
```

```
()
```

```
def squares(n = 10):
    print(f"Generating squares from 1 to {n **2}")
    for i in range (1, n + 1):
        yield i ** 2
```

```
gen = squares()
```

```
gen
```

```
<generator object squares at 0x00000193C6733120>
```

```
# use for loop to see the output from generator
for x in gen:
    print(x, end= ' ')
```

```
Generating squares from 1 to 100
1 4 9 16 25 36 49 64 81 100
```

Generator expressions

```
gen2 = (x ** 2 for x in range(100))

gen2
```

```
<generator object <genexpr> at 0x00000193C6733D60>
```

```
for x in gen2:
    print(x, end= " ")
```

```
0 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625 676
```

```
# verbose generator

def _make_gen():
    for x in range(100):
        yield x** 2

gen = _make_gen()
```

```
# using gen expressions as function arguments

sum (x ** 2 for x in range(100))
```

```
328350
```

```
dict((i, i**2 ) for i in range(5))
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

itertools module

- contains collection of generators for many data algorithms

```
import itertools

def first_letter(x):
    return x[0]

names =  ['Alan', 'Adam', 'Wes', 'Will', 'Albert']

for letter, names in itertools.groupby(names, first_letter):
    print(letter, list(names))
```

```
A ['Alan', 'Adam']
W ['Wes', 'Will']
A ['Albert']
```

Errors and exception handling

```
def attempt_float(x):
    try:
        return float(x)
    except:
        return x
```

```
1.22
```

```
attempt_float('flower')
```

```
'flower'
```

```
def attempt_float1(x):
    try:
        return float(x)
    except(TypeError, ValueError):
        return x
```

```
attempt_float1('nice')
```

```
'nice'
```