

Practice exercises: functions

The FOSSEE Group

Department of Aerospace Engineering
IIT Bombay

Mumbai, India

Note: Python 2.x and 3.x

If you are using Python 2.x

- Use `raw_input` instead of `input`
- Use the following for `print`

```
from __future__ import print_function
```

Exercise: function without arguments

- 1 Define a function called **prompt**
- 2 The function should ask the user to enter their name (no prompt)
- 3 The function should not take any arguments
- 4 The function should not return anything
- 5 The function should print **Hello <name>**

For example if the user enters **Sam**, print: Hello Sam

Solution

```
def prompt():  
    name = input()  
    print('Hello', name)
```

Exercise: function with one argument

- 1 Define a function called **prompt**
- 2 The function should take a single string argument
- 3 **Do not ask the user for input, i.e. do not use input**
- 4 The function should not return anything
- 5 The function should print **Hello <name>**

For example if the function is passed ' **Sam** ' , print:
Hello Sam

Solution

```
def prompt(name):  
    print('Hello', name)
```

Exercise: function with **return**

- 1 Define a function called **prompt**
- 2 The function should take a single string argument
- 3 The function should return a string with '**Hello**'
- 4 **Do not use input**
- 5 **Do not print anything**

For example if the function is passed '**Sam**', return:

'Hello Sam'

Solution

```
def prompt (name) :  
    return 'Hello ' + name
```


Exercise: function with two args

- 1 Define a function called **add**
- 2 The function should take two arguments
- 3 The function should return the sum of the two arguments
- 4 **Do not use input**
- 5 **Do not print anything**

Solution

```
def add(a, b):  
    return a + b
```

Exercise: function returning boolean

- 1 Define a function called **is_even**
- 2 The function should take a single integer argument
- 3 The function should return **True** if the number is even and **False** otherwise
- 4 **Do not use input**
- 5 **Do not print anything**

Naive Solution

```
def is_even(x):  
    if x%2 == 0:  
        return True  
    else:  
        return False
```

Elegant Solution

```
def is_even(x):  
    return x%2 == 0
```

Exercise: function returning two values

- 1 Define a function called **even_square**
- 2 The function should take a single argument
- 3 The function should return if the number is even and the square of the number
- 4 **Do not use input**
- 5 **Do not print anything**

For example:

```
In []: even_square(2)
```

```
Out []: (True, 4)
```

```
In []: even_square(3)
```

```
Out []: (False, 9)
```

Solution

```
def even_square(x):  
    return x%2 == 0, x*x
```

Exercise: default arguments

- 1 Define a function called **greet**
- 2 The function should take one positional argument, **name**
- 3 The function should take one optional argument, **message**
- 4 If **message** is not given, it should default to **'Hello'**
- 5 It should return the string with the greeting

For example:

```
In []: greet('Sam')
```

```
Out []: 'Hello Sam'
```

```
In []: greet('Sam', 'Hi')
```

```
Out []: 'Hi Sam'
```


Solution

```
def greet(name, message='Hello') :  
    return message + ' ' + name
```

Exercise: functions and lists

- 1 Define a function called **to_lower**
- 2 The function should take a single list of strings
- 3 The function should return the list of strings but all in lowercase
- 4 **Do not use input**
- 5 **Do not print anything**

For example:

```
In []: to_lower(['I', 'am', 'Batman'])  
Out []: ['i', 'am', 'batman']
```

Solution

```
def to_lower(data):  
    result = []  
    for x in data:  
        result.append(x.lower())  
    return result
```

Exercise: list of Fibonacci

- 1 Define a function called **fib** taking one argument **n**
- 2 Where, **n>0** is an integer but defaults to 8
- 3 Return the first **n** terms of the Fibonacci sequence

For example:

```
In []: fib(4)
```

```
Out []: [0, 1, 1, 2]
```

```
In []: fib()
```

```
Out []: [0, 1, 1, 2, 3, 5, 8, 13]
```

Solution

```
def fib(n=8):  
    a, b = 0, 1  
    result = [0]  
    for i in range(n-1):  
        result.append(b)  
        a, b = b, a+b  
    return result
```

Exercise: returning a function

- 1 Define a function called **power2 ()** which takes no argument
- 2 It should return a function which takes a single argument **x** but returns 2^x

For example:

```
In []: f = power2 ()
```

```
In []: f (2)
```

```
Out []: 4
```

```
In []: power2 () (4)
```

```
Out []: 16
```

Solution

```
def power2():  
    def f(x):  
        return 2**x  
    return f
```

Another solution

```
def power(n=2):  
    def f(x):  
        return n**x  
    return f
```

- This is called a closure.
- Note that `f` “stores” the value of `n`

```
In []: p2 = power(2)
```

```
In []: p3 = power(3)
```

```
In []: p2(2)
```

```
Out []: 4
```

```
In []: p3(2)
```

```
Out []: 9
```


Another solution

```
def power(n=2):  
    def f(x):  
        return n**x  
    return f
```

- This is called a closure.
- Note that `f` “stores” the value of `n`

```
In []: p2 = power(2)
```

```
In []: p3 = power(3)
```

```
In []: p2(2)
```

```
Out []: 4
```

```
In []: p3(2)
```

```
Out []: 9
```

Exercise: function as an argument

- 1 Define a function called `apply(f, data)`
- 2 Where `f` is a function taking a single value
- 3 Where `data` is a list
- 4 It should return a list where the function is applied to each element of `data`

For example:

```
In []: def double(x):  
.....:     return 2*x  
.....:  
In []: apply(double, [1, 2, 3])  
Out []: [2, 4, 6]
```

Solution

```
def apply(f, data):  
    result = []  
    for x in data:  
        result.append(f(x))  
    return result
```

That's all folks!