Functions p.II
Advanced

Exercises

Level - Easy

Exercise 1-1

Write a function sum_all that takes **any number of arguments** and returns their sum. The function should also handle the case where no arguments are passed and return a suitable message.

sum_all(1, 2, 3) should returns 6
sum_all(10, 20) should returns 30

Exercise 1-2

Create a function count_args that returns the **total number of arguments** passed to it. The function should also handle the case where no arguments are passed and return a suitable message.

count_args(1,2,3) should returns 3
count_args(“a”, 2, 3.14, True) should returns 4

Exercise 1-3

Write a function concat_strings that concatenates all arguments and returns the combined string. The function should also handle the case where no arguments are passed and return a suitable message.

concat_strings("Hello", " ", "World!") should return 'Hello World!'
concat_strings("Is", "Python", " fun", "?") should return “Is Python fun ?” (and not “no”)

Exercise 1-4

Write a function filter_by_type that goes through the named arguments and returns a dictionary where the keys are types (int, str, etc.) and the values are lists of the items of that type using kwargs.

filter_by_type(name="Alice", age=30, city="New York", score=85.5) should return {str: ["Alice", "New York"], int: [30], float: [85.5]}

Exercise 1-5

Write a function update_dict which takes a dictionary original and updates it with any other named arguments passed using kwargs.

Exercise 1-6

Write a lambda function that takes two arguments and returns their sum.

Level - Moderate

Exercise 2-1

Write a function multiply_and_sum that has a multiplier argument and can accept any number of other arguments. It should return the sum of the other arguments multiplied by the multiplier argument.The function should also handle the case where no arguments are passed and return a suitable message.

Exercise 2-2

Write a lambda function to sort the following list of tuples by the second item in each tuple:

[(5,3),(2,2),(4,7),(0,1),(1,6)]

Exercise 2-3

Write a lambda function with the filter() function to extract even numbers from the following list:

[1,2,3,4,5,6,7,8,9]

Exercise 2-4

Write a lambda function with the filter function to find the intersection of the two following lists :

[0,3,5,6,7,8,10]
[1,3,4,5,7,8]

Exercise 2-5

Write a lambda function with the map() function to square all elements of the following list:

[1, 2, 3, 4, 5]

Exercise 2-6

Write a function that adds the two given lists using the map builtin function and lambda.

l1 = [1, 2, 3]
l2 = [4, 5, 6]
Expected result → [5, 7, 9]

Exercise 2-7

Write a function greet(name) that contains a nested function message() which returns a greeting message with the specified name. The greet() function should return the result of calling message().

greet("Alice")) should return "Hello, Alice!"

Exercise 2-8

Create a function multiplier(n) which returns a nested function. This nested function should take a number and return it multiplied by n.

if double = multiplier(2), double(5) should return 10
if triple = multiplier(3), triple(3) should return 9

Exercise 2-9

Write a function create_counter() that returns a nested function. The nested function should count how many times it has been called and return this number.

counter = create_counter()
print(counter()) should return 1
print(counter()) should return 2
print(counter()) should return 3

Level - Hard

Exercise 3-1

Write a lambda function to sort the following list of dictionaries based on the model key:

```
[{'model': 7, 'color': 'Black'}, {'model': 2, 'color': 'Gold'},
 {'model': 7, 'color': 'Blue'}, {'model': 2, 'color': 'White'},
 {'model': 3, 'color': 'Yellow'}, {'model': 3, 'color': 'Pink'}]
```

Modify this function to sort it first by model, then by color. It should only use one line of code.

Exercise 3-2

Write a function to sort the numbers and then put the negative number in front of the positive numbers using Lambda.

if mylist = [-1, 2, -3, 5, 7, 8, 9, -10] then the result should be [2, 5, 7, 8, 9, -10, -3, -1]

Exercise 3-3

Write a password generator consisting of two nested functions:

- An **outer function** responsible for defining the set of characters to be used in the password. This function can take a list where each element is a string representing a set of characters. *For example, ['abcdefg', '123456789'] represents two sets of characters - one alphabetical and one numerical – that will be used to generate the password.*
- An **inner function** that generates the password using the character sets provided by the outer function. It should take a length parameter that specifies the desired length of the password.The function randomly selects characters from the combined character sets provided to the outer function to construct a password of the specified length.

To randomly select the characters you can use the random module:

```
import random
character = random.choice(characters)
```

where characters is a list/string that contains all of the available characters