

Dynamic Modelling Course - TEP4290: Warm-up 5

The point of this exercise is for you to practice what you have learned in the recommended videos and notebook for custom functions:

- PY4E: <https://www.py4e.com/lessons/functions#>
- Whirlwind tour of python: <https://jakevdp.github.io/WhirlwindTourOfPython/08-defining-functions.html>

You will perform some basic operations that are designed to help you get comfortable with functions in Python. The exercise is pass/fail and contributes to the required 8/12 warm-ups you need to pass.

Good luck!

Quick summary

Built-in vs self defined functions

We can call functions in python that then do stuff for us. This can be built-in functions such as `print()`, or custom built functions like you will create here.

For built-in functions it can be helpful to look up their documentation to see what they can do - for example for `print()` : <https://docs.python.org/3/library/functions.html#print>. Here you can see that `print` can print multiple arguments with different separators we can specify

```
In [1]: #example what print() can do:
print(1, 2, 3)
#vs
print(1, 2, 3, sep = '-')
#vs
print(1,2,3, sep = ' words ')
```

```
1 2 3
1-2-3
1 words 2 words 3
```

Custom functions can be a good way to avoid repeating code, but also to break up difficult and complex tasks into much simpler ones, while maintaining a quick and userfriendly code (similar to what object-oriented programming offers).

Anatomy of a python function definition

In python we define such functions like this:

name of the function:
avoid capital letters

argument (input):
use mnemonic names
and think about what
type of input you want
(str, int, float, lists)

keyword argument (input):
argument with a default value, if not
specified, default will be used, needs to
come after regular arguments

definition `def my_function (argument1, keyword_argument1 = 'default1'):`

docstring `"""This function is an example"""` ← comment that documents the function

body

`my_result = argument1 + keyword_argument1` ← actional functionality:
use the arguments to get
the desired results

`return my_result` ← return statement:
terminates function and hands result

indentation after
the definition

`my_result = my_function('my_string', keyword_argument1 = 'default2')`

Example call:
my_result will store a string, 'my_stringdefault2'

Note that a function can take multiple or no arguments, and return multiple or no results.

Here a simple example for a function that does not return any results:

```
In [2]: def hello_name(name):
        print('Hello ' + name)
        return

        hello_name('Daniel')
```

Hello Daniel

And here one that implements insertion sort to sort a list of numbers. You do not need to understand the code within the function (the sorting algorithm), only what arguments it needs and what it returns

```
In [3]: def sort_numbers(list_of_numbers):

        # Traverse through 1 to len(list_of_numbers)
        for i in range(1, len(list_of_numbers)):

            key = list_of_numbers[i]

            # Move elements of arr[0..i-1], that are
            # greater than key, to one position ahead
            # of their current position
            j = i-1
            while j >= 0 and key < list_of_numbers[j] :
                list_of_numbers[j+1] = list_of_numbers[j]
                j -= 1
            list_of_numbers[j+1] = key
        return list_of_numbers

        test_list = [ 3, 5, 1, 7, 6]

        ordered_list = sort_numbers(test_list)
        print(ordered_list)
```

[1, 3, 5, 6, 7]

Tasks

Complete the tasks outlined below to achieve the same output as you find in the original file. Use the specific method described if applicable.

Complete add function

Complete the function add that returns the sum of two integers

```
In [4]: def add(a,b):  
        sum = a + b  
        return sum  
  
print(add(2,2))  
print(add(1024, 2048))
```

4
3072

Complete greeting function

Fill in code within the function below using if/elif/else statements.

```
In [5]: def greet_teacher(teacher):  
        '''  
        Prints a greeting in the native language of the teacher.  
  
        Arguments:  
        teacher : string with the name of the teacher  
  
        Returns:  
        -  
        ...  
  
        if teacher == 'Fernando':  
            greeting = 'Hola'  
        elif teacher == 'Marceau':  
            greeting = 'Bonjour'  
        elif teacher == 'Daniel':  
            greeting = 'Hallo'  
        else:  
            print(teacher + ' is not a teacher!')  
            return  
  
        print(greeting, teacher)  
  
        return
```

```
greet_teacher('Fernando')
greet_teacher('Marceau')
greet_teacher('Daniel')
greet_teacher('Michael Jackson')
```

```
Hola Fernando
Bonjour Marceau
Hallo Daniel
Michael Jackson is not a teacher!
```

Payroll function

Write a function to compute the gross pay of a worker. Pay should be the normal rate for hours up to 40 and time-and-a-half for the hourly rate for all hours worked above 40 hours. Put the logic to do the computation of pay in a function called `payroll()` and use the function to do the computation. The function should return a value.

Use 45 hours and a rate of 10.50 per hour to test the function (the pay should be 498.75). Do not name your variable `sum` or use the `sum()` function.**

```
In [6]: def payroll(hours, rate):
        if hours <= 40:
            gross_pay = hours * rate
        else:
            gross_pay = 40 * rate + (hours - 40) * rate * 1.5

        return gross_pay

payroll(45, 10.5)
```

```
Out[6]: 498.75
```

Maximum value

You can reuse functions you already defined to make your work on new functions easier. Here you will use the `sort_numbers` function that we defined above to create a function called `maximum_value` that returns the highest value from a list of numbers. You **must** use the `sort_numbers` function!

Test your function with the lists `[4, 2, 9, 8]` and `[1, -1, 1000, 0.5]`

```
In [7]: def maximum_value(list_of_numbers):
        sort_list = sort_numbers(list_of_numbers)
        return sort_list[-1]

print(maximum_value([4,2,9,8]))
print(maximum_value([1,-1,1000,0.5]))
```

9
1000

Best job

You will now make use of the two last functions you defined to make a function called `best_job` and takes in 4 values describing two jobs (a rate for the hourly pay in each job and a number of hours for each job) and returns the payroll for the higher paying job. You must make use of the `maximum_value` and `payroll` function.

Test your function with one job with 50 hours and a rate of 190 kr/hour and one with 35 hours and 225 kr/hours.

```
In [9]: def best_job(job1_hours, job1_rate, job2_hours, job2_rate):  
        job1_pay = payroll(job1_hours, job1_rate)  
        job2_pay = payroll(job2_hours, job2_rate)  
  
        list_of_pays = [job1_pay, job2_pay]  
        max_pay = maximum_value(list_of_pays)  
  
        return max_pay  
  
jop1_hours = 50  
jop1_rate = 190  
jop2_hours = 35  
jop2_rate = 225  
print(best_job(jop1_hours, jop1_rate, jop2_hours, jop2_rate))
```

10450.0

Well done!