Warm_up_05_function_definitions

January 12, 2025

1 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# - Whirwind 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 comforable with functions in Python. The exercise is pass/fail and contributes to the required 8/12 warm-ups you need to pass.

Good luck!

1.0.1 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 seperators we can specify

[3]:
```python
#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:

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:

[4]:
```
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

[5]:
```
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]
```

2 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.

2.0.1 Complete add function

Complete the function add that returns the sum of two integers

[5]:
```
def add(a,b):
    result = a + b
    return result
```

```
print(add(2,2))
print(add(1024, 2048))
```

```
4
3072
```

2.0.2 Complete greeting function

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

[8]:
```
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'
    #add content
    elif teacher == 'Marceau':
        greeting = 'Bonjour'

    elif teacher == 'Daniel':
        greeting = 'Hallo'

    elif teacher == 'Michael Jackson':
        greeting = 'Hello'

    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
Hello Michael Jackson
```

2.0.3 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.

[35]:
```python
def payroll(hours_worked, hourly_rate):
    if hours_worked <= 40:
        gross_pay = hours_worked * hourly_rate
    else:
        regular_pay = 40 * hourly_rate
        overtime_pay = (hours_worked - 40) * (hourly_rate * 1.5)
        gross_pay = regular_pay + overtime_pay
    return gross_pay

#test with 45 hours and a rate of 10.50
hours = 45
rate = 10.50
pay = payroll(hours, rate)
print(f'The gross pay for {hours} hours at a rate of {rate} per hour is {pay}.')
```

```
The gross pay for 45 hours at a rate of 10.5 per hour is 498.75.
```

2.0.4 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]

[36]:
```python
def maximum_value(numbers):
    sorted_numbers = sort_numbers(numbers)
    return sorted_numbers[-1]

# Test the function with the lists [4, 2, 9, 8] and [1, -1, 1000, 0.5]
test_list1 = [4, 2, 9, 8]
test_list2 = [1, -1, 1000, 0.5]

print(f'The maximum value in {test_list1} is {maximum_value(test_list1)}.')
print(f'The maximum value in {test_list2} is {maximum_value(test_list2)}.')
```

```
The maximum value in [4, 2, 9, 8] is 9.
The maximum value in [1, -1, 1000, 0.5] is 1000.
```

2.0.5 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.

[39]:
```
def best_job(hours1, rate1, hours2, rate2):
    pay1 = payroll(hours1, rate1)
    pay2 = payroll(hours2, rate2)
    return maximum_value([pay1, pay2])

# Test the function with one job with 50 hours and a rate of 190 kr/hour and␣
 ↪one with 35 hours and 225 kr/hour
hours_job1 = 50
rate_job1 = 190
hours_job2 = 35
rate_job2 = 225

best_pay = best_job(hours_job1, rate_job1, hours_job2, rate_job2)
print(f'The higher paying job has a payroll of {best_pay} kr.')
```

```
The higher paying job has a payroll of 10450.0 kr.
```

3 Well done!