

# 15-110 Fall 2019

## Hw 02

Out: Saturday 7<sup>th</sup> September, 2019 at 18:30 AST

Due: Thursday 12<sup>th</sup> September, 2019 at 17:30 AST

### Introduction

In this homework you will practice with reading, understanding, and writing basic python code.

**The total number of points available from the questions is 100.**

In your `.zip` file (see general instructions below), you need to include two files: (i) `hw01.pdf`, with the answers to the questions. You can use either a pen or a PDF editor to type your answers (on this same file!). (ii) `hw02.py`, with the python functions answering the questions in Section 2. Watch out: you must write your functions *both* in the `hw02.py` file and in the `hw02.pdf` file.

### General Instructions for Submitting the Assignments

Submissions are handled through Autolab, at <https://autolab.andrew.cmu.edu/courses/15110q-f19>

You are advised to create on your computer/account a folder named `110-hw`. For each new homework, you should create a new sub-folder named `01`, `02`, etc. where you can put the files related to the homework. In this way your work will be nicely organized and information and files will be easily accessible.

You can also create an equivalent structure for the *laboratories*, where in this case the root folder should be named `110-lab`.

When you are ready with the homework and want to submit your solutions, you need to go in the current homework folder (e.g., `01`), *select all files you will submit* (that can include both `.pdf` files with written answers to questions and python code files, `.py`) and compress them in one single `.zip` file.<sup>1</sup>

According to the OS you are using, you might have different options for making the zip file. For instance, on Windows, after selection of the files, you should right-click and select **Send to: Compressed folder**, while on macOS, you can select **Compress** on the menu appearing from the right-click.

The compression action will produce a zip file containing the files to be handed in for the assignment. The file should be named `hwXX-handin.zip` (e.g., for this homework, the name of the file should be `hw02-handin.zip`). Then, open [Autolab](#), find the page for this assignment, and submit your `hw02-handin.zip` file via the “Submit” link.

☛ The number of submissions is limited to 5. The last submission is the one that will be graded.

### Style

Part of your grade on assignments are style points, that can be lost if your code is too disorganized, unreadable or unnecessarily complicated. To avoid losing style points, please follow the guidelines at <https://web2.qatar.cmu.edu/cs/15110/resources/style.pdf>.

---

<sup>1</sup>The (single) zip file is needed, even when the files handed for the assignment consists of one individual file.

# 1 Read and understand code

1. **6 points** The following piece of code should output the value of `x` if `x` has a value strictly less than 3, and should output `Hello!` otherwise. Unfortunately, the code has a few syntactic and semantic errors: spot them and write the corrected version of the code.

```
x = 2
y = 'Hello!' + x
if x > 3
    print(y)
else
    Print(x)
```

2. Let's consider the following lines of code:

```
y = x % 10
y = 10 * y
if y < 0 or y >= 10:
    y = y * 100 // x
elif y > 1 and y < 10:
    y = 22
else:
    y = x
print(y)
```

- (a) **5 points** What is the output of the code if we first perform the assignment `x = 100`?

- (b) **5 points** How would you rewrite the second line (`y = 10 * y`) such that for `x = 101` the output becomes 22? (note: the new expression for the assignment to `y` must still include `x`, e.g., `y = x + 2`)

- (c) **5 points** In the original code, add an additional `elif` condition on `y` such that the output becomes I am a line of `code`! for `x = 200`.

3. **6 points** The purpose of the following lines of code is to convert temperature degrees from Celsius to Fahrenheit and print out the result. If the Celsius temperature is over 50 degrees a warning should be printed out, reporting the temperature in Fahrenheit. Otherwise, the temperature according to the two scales should be printed out. The Fahrenheit temperature needs to be printed out always as an integer value. Complete the code to get no errors and obtain the desired result. You need to fill in the sections indicated by `----` as well as other parts of the code that might need corrections. Note that the symbol `' '` in the code means a white space.

```
celsius = 41.3
fahrenheit = ----- * (9/5) + 32
if celsius > 50
    print("It's hot out of there!", -----, "Fahrenheit degrees")
else
    Print("Temperature:", -----, "(C)", -----, "(F)")
```

4. The function `input(msg)`, that can be used to read an input from the keyboard, returns a `string` object. Let's assume that as an input we have given a character corresponding to an integer (e.g., `'5'`), and that the string object returned by `input()` has been stored in the string variable `v`.

- (a) **6 points** Which ones of the lines of code below would achieve the result of transforming `v` into a variable of integer type? (note that value could be changed compared to original input).

- ☐ `v = v(int)`
- ☐ `v = int(v)`
- ☐ `v = Int(v)`
- ☐ `v = int(v) + 0.0`
- ☐ `v = int(v) + 0`
- ☐ `v = bool(v)`
- ☐ `v = bool(v) + 0`
- ☐ `v = v + 0`

- (b) **5 points** Is the assignment `x = int(v)` a type conversion or a type casting operation?

- ☐ Conversion
- ☐ Casting

(c) **5 points** Given that `v` is currently a string object, the assignment `v = 3` would result into an operation of:

- ☐ Type conversion
- ☐ Type Casting
- ☐ None of the two above

5. **6 points** What is the minimal value that the integer variable `x` can take in order to let the following code printing out `third line`?

```
if x > 10 and x < 20:
    print('first_line')
elif x == 20:
    print('second_line')
elif x // 20 >= 2 and x // 5 > 10:
    print('third_line')
else:
    print("I don't know!")
```

6. **6 points** Given the code below, what are integer values that should be assigned to the variables `x` and `y` such that `z` gets assigned the value 1001? Check the correct answers.

```
z = id(x) + id(y)
```

- ☐ `x = 1000, y = 1`
- ☐ `x = 1, y = 1000`
- ☐ `x = 500, y = 1001 - x`
- ☐ No way to ensure that `z = 1001` because the integer variables `x` and `y` are mutable types.
- ☐ No way to ensure that `z = 1001` because the integer variables `x` and `y` are immutable types.
- ☐ No way to ensure that `z = 1001` because the memory addresses where the values associated to the variables `x` and `y` are stored are managed by the python interpreter.
- ☐ None of the above answers.

## 2 Write python code

1. **10 points** Write the function `multiple_of_seventeen(n)` that takes as input an integer `n` and returns `True` if `n` is a multiple of 17 and `False` otherwise.

- Example cases for the function returning `True`: `n = 34, 68, 17`.
- Example cases for the function returning `False`: `n = 33, 67, 16, 1`.

2. **15 points** Write the function `even_or_multiple_of_five_and_thirteen(n)` that takes as input an integer and returns `True` if this integer is a multiple of 5 and 13 (both), or if it is even. Return `False` otherwise.
- Example cases for the function returning `True`: `n = 65, 68, 130, 195, 2`.
  - Example cases for the function returning `False`: `n = 5, 13, 31, 27, 1, 193`.

3. **20 points** Write the function `max_two(a, b)` that takes two objects and returns the greatest of the two. If they are the same, return either of them. If they cannot be compared, because one of the two inputs is a string and the other is not, return the string `'Error'`.
- Example cases, in the form of pairs `([a, b], returned_value)`:
    - `([1, 4], 4)`, `([9.4, 7], 9.4)`, `(['a', 'b'], 'b')`
    - `([True, False], True)`, `(['Hello', 5], 'Error')`, `([True, 5], 5)`

## Appendix: How to define, use, and test custom functions

In the questions in Section 2 of this homework, and in most of the questions of future homework, you are asked to write a **function** that generates the required output. Functions were briefly introduced in the class (Lecture 3), mainly to describe the use of Python's *built-in* functions such as `print()`, `id()`, `type()`, `int()`, etc. Here we summarize the basic syntax for writing **user-defined** functions (i.e., functions that the programmer writes for doing useful things) and we show simple examples for using and testing such functions in the Anaconda environment.

### Function definition: creating a new function

A user function needs to be **defined** using the keyword `def`. A function always has a unique **name**, suitably chosen by the user, that can be used to refer to the function object anywhere in the code. In the very general example below, the function has been named `function_name`.

```
def function_name(arg1, arg2, arg3):  
    body_of the function  
    return some_values
```

A function can have (or not) **input arguments**. In the example, the function has three arguments that are named `arg1`, `arg2`, `arg3`. The arguments are treated as **variables** inside the body of the function.

Similarly to `if/else` statements, a *colon* symbol `:` ends the first line of the function definition. All the lines of code that are **indented** w.r.t the `def` keyword represent the **body of the function**, the code that generates the desired outputs (based on the given inputs).

A function can (or not) **return objects**. This is achieved by the keyword `return`. In the example, the object `some_values` is returned. In general, zero, one, or multiple objects (separated by commas) can be returned.

### Use of a function

The following code defines a function that computes the sum of two numbers, prints out the result, and returns the computed sum:

```
def sum_two_numbers(x, y):  
    z = x + y  
    print('The sum is: ', z)  
    return z
```

You may notice that in the code above, the input arguments `x` and `y` are variables (objects with a name) in the function's body, meaning that they can be assigned to any value. The precise value that they take is only specified when the function is being **invoked** (*called*) somewhere in the program.

For example, once defined, the function `sum_two_numbers(x, y)` can be used inside the piece of code below, where the function is first invoked passing two integer literals, 2 and 3. Then, in the third line the function is invoked again using two external variables as arguments. The effect is that inside the function the internal variables `x` and `y` take on different values. The value of `total` in the example is 11.

```
a = sum_two_numbers(2,3)  
b = 6  
total = sum_two_numbers(a, b)
```

☛ Note that a function can both return values *and* print out results, as in the example. These are two different types of output. Returned values are meant to be used by the code invoking the function, while printed out results can serve multiple purposes (e.g., to track what's going on, or to report partial results of a computation).

## How to answer to a typical homework question?

A typical homework question looks like the following.

*Write the function `product(a,b,c)` that takes as input three integer variables. The function returns the value of the product of the three variables as a float and prints out the string `'The product is even'` if the resulting product is an even number, or the string `'The product is odd'` if the result is odd.*

In your answer, you must follow the given specifications **exactly**. In this case you should likely write the following function, that returns a float object and produces a print, precisely as required.

```
def product(a,b,c):
    p = a * b * c
    if (p % 2) == 0:
        print('The product is even')
    else:
        print('The product is odd')
    return float(p)
```

☛ Returned types/values that are wrong, extra spaces, spelling errors, wrong formatting, will all count as **errors**!. You'll be graded mainly on the correctness of the outputs, but also on how *good* your code is in terms of **style** and **efficiency** (no unnecessary, CPU- or memory-consuming operations should be there).

For instance, the following function achieves the same result of the previous one, but is written in a very bad style (no self-explanatory names for the variables, no spaces between operators and operands), and is much more convoluted and inefficient of the previous one (a lot of unnecessary variables and operations, not using proper built-in functions):

```
def product(a,b,c):
    p = 1
    p1 = p * a
    p2 = p1*b
    p3 = p2*c
    zz = p3%2
    if zz==0:
        print('The product is even')
    else:
        print('The product is odd')
    zzz=p3 + 0.0
    return zzz
```

**Moral:** learn since the beginning how to write clean and efficient code, with minimal redundancies!

## How do you test your function?

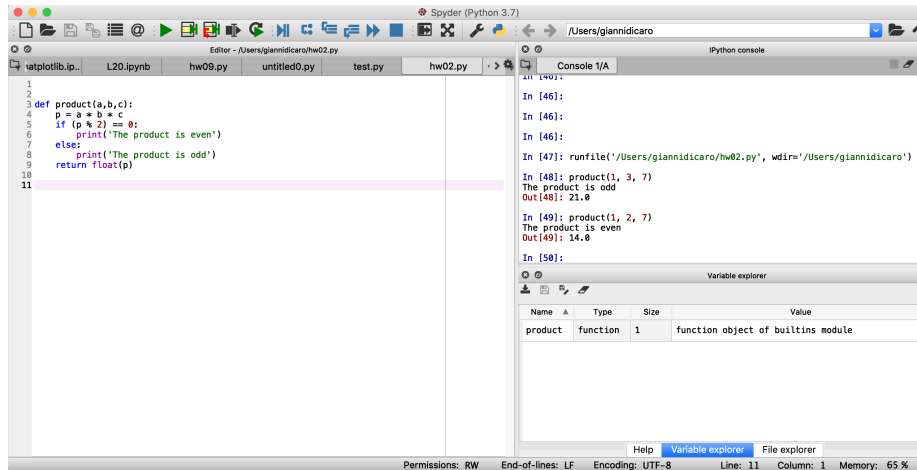
You have written your function: how do you check that it does the job precisely as specified? This step is extremely important in programming! You need to **test your function on different inputs** and ensure that the outputs are always compliant with the given requirements.


In practice, you should perform tests both mentally, maybe with pen and paper, and using the computer. The **Spyder** IDE offers flexible ways to do it using both the console and the file editor tools.

- **Test the defined function directly using the *Spyder console*.**

In the *Spyder console*, you can invoke the function with different inputs and check the outputs.

The figure shows an example screenshot taken from Spyder. In the left window (the **file Editor**) we wrote the function `product()`.



Once written, the code of the function needs to be **executed**, which is achieved by using the *Run* button . After execution, if there are no errors, the function object named `product` is now *defined* in our environment and can be used.

In the right-top window (the **Interactive Console**), the now defined function is invoked from the command-line, first with arguments 1, 3, 7 (In [48]) and then with arguments 1, 2, 7 (In [49]). The results (after executing each command by pressing the *Return* button) are shown in the Console.

Keep in mind that every time you change the function in the file Editor, you need to re-run it (push again the *Run* button), in order to update its definition.

Note that the right-bottom window (**Variable Explorer**), shows that only one variable, the named function object `product`, is being defined in the programming environment.

- **Test the defined function by invoking it inside other code.** Another way to perform tests is by adding lines of code with test inputs directly in the file where the function has been defined. When the file with the code is executed (as described above), these lines of code will be also executed (sequentially) and the results will be shown in the console.

This way of proceeding, which is more appropriate for extensive testing, is shown in the example screenshot below. In this case, the function is being tested with many different inputs.

