# ClassPractice-Functions

October 7, 2024

## 1 In Class Practice: Functions

The goals of this lab are to help you to understand:

- How to use Python's built-in functions in the standard library.
- How to write user-defined functions
- How to use other people's code.
- The benefits of user-defined functions to code reuse and simplicity.
- How to create a program to use functions to solve a complex idea

We will demonstrate these through the following example:

### 1.1 The Credit Card Problem

If you're going to do commerce on the web, you're going to support credit cards. But how do you know if a given number is valid? And how do you know which network issued the card?

**Example:** Is `5300023581452982` a valid credit card number?Is it? Visa? MasterCard, Discover? or American Express?

While eventually the card number is validated when you attempt to post a transaction, there's a lot of reasons why you might want to know its valid before the transaction takes place. The most common being just trying to catch an honest key-entry mistake made by your site visitor.

So there are two things we'd like to figure out, for any "potential" card number:

- Who is the issuing network? Visa, MasterCard, Discover or American Express.
- IS the number potentially valid (as opposed to a made up series of digits)?

#### 1.1.1 What does this have to do with functions?

If we get this code to work, it seems like it might be useful to re-use it in several other programs we may write in the future. We can do this by writing the code as a **function**. Think of a function as an independent program its own inputs and output. The program is defined under a name so that we can use it simply by calling its name.

**Example:** `n = int("50")` the function `int()` takes the string `"50"` as input and converts it to an `int` value `50` which is then stored in the value `n`.

When you create these credit card functions, we might want to re-use them by placing them in a **Module** which is a file with a collection of functions in it. Furthermore we can take a group of related modules and place them together in a Python **Package**. You install packages on your computer with the `pip` command.

## 1.2 Built-In Functions

Let's start by checking out the built-in functions in Python's math library. We use the `dir()` function to list the names of the math library:

[ ]:
```
import math
dir(math)
```

If you look through the output, you'll see a `factorial` name. Let's see if it's a function we can use:

[ ]:
```
help(math.factorial)
```

It says it's a built-in function, and requies an integer value (which it referrs to as x, but that value is arbitrary) as an argument. Let's call the function and see if it works:

[ ]:
```
math.factorial(5) #this is an example of "calling" the function with input 5.␣
↪The output should be 120
```

[ ]:
```
math.factorial(0) # here we call the same function with input 0. The output␣
↪should be 1.
```

### 1.2.1 1.1 You Code

Call the factorial function with an input argument of 4. What is the output?

[ ]:
```
#TODO write code here.
```

Now let's use these two functions in a familiar program, along with `interact_manual()` to make the inputs as awesome as the outputs!

## 1.3 Get Down with OPC (Other People's Code)

Now that we know a bit about **Packages**, **Modules**, and **Functions** let me expand your horizons a bit. There's a whole world of Python code out there that you can use, and it's what makes Python the powerful and popular programming language that it is today. All you need to do to use it is *read*!

For example. Let's say I want to print some emojis in Python. I might search the Python Package Index `https://pypi.org/` for some modules to try.

For example this one: https://pypi.org/project/emoji/

Let's take it for a spin!

### 1.3.1 Installing with pip

First we need to install the package with the `pip` utility. This runs from the command line, so to execute pip within our notebook we use the bang `!` operator.

This downloads the package and installs it into your Python environment, so that you can `import` it.

[ ]:
```
!pip install emoji
```

Once the package is installed we can use it. Learning how to use it is just a matter of reading the documentation and trying things out. There are no short-cuts here! For example:

[ ]:
```
# TODO: Run this
import emoji
print(emoji.emojize('Python is :thumbs_up:'))
print(emoji.emojize('But I thought this :lab_coat: was supposed to be about :
 ↪credit_card: ??'))
```

### 1.3.2 1.2 You Code

Write a python program to print the bacon, ice cream, and thumbs-down emojis on a single line.

[ ]:
```
## TODO: Write your code here
```

## 1.4 Let's get back to credit cards....

Now that we know a bit about **Packages**, **Modules**, and **Functions** let's attempt to write our first function. Let's tackle the easier of our two credit card related problems:

- Who is the issuing network? Visa, MasterCard, Discover or American Express.

This problem can be solved by looking at the first digit of the card number:

- "4" ==> "Visa"
- "5" ==> "MasterCard"
- "6" ==> "Discover"
- "3" ==> "American Express"

So for card number `5300023581452982` the issuer is "MasterCard".

It should be easy to write a program to solve this problem. Here's the algorithm:

```
input credit card number into variable card
get the first digit of the card number (eg. digit = card[0])
if digit equals "4"
    the card issuer "Visa"
elif digit equals "5"
    the card issuer "MasterCard"
elif digit equals "6"
    the card issuer is "Discover"
elif digit equals "3"
    the card issues is "American Express"
else
    the issuer is "Invalid"
print issuer
```

### 1.4.1 1.3 You Code

Debug this code so that it prints the correct issuer based on the first card

[ ]:
```
## TODO: Debug this code
card = input("Enter a credit card: ")
digit = card[0]
if digit == '5':
    issuer = "Visa"
elif digit == '5':
    issuer == "Mastercard"
elif digit = '6':
    issuer == "Discover"
elif digit == '3'
    issuer = "American Express"
else:
    issuer = "Invalid"
print(issuer)
```

**IMPORTANT** Make sure to test your code by running it 5 times. You should test issuer and also the "Invalid Card" case.

## 1.5 Introducing the Write - Refactor - Test - Rewrite approach

It would be nice to re-write this code to use a function. This can seem daunting / confusing for beginner programmers, which is why we teach the **Write - Refactor - Test - Rewrite** approach. In this approach you write the ENTIRE PROGRAM and then REWRITE IT to use functions. Yes, it's inefficient, but until you get comfotable thinking "functions first" its the best way to modularize your code with functions. Here's the approach:

1. Write the code
2. Refactor (change the code around) to use a function
3. Test the function by calling it
4. Rewrite the original code to use the new function.

We already did step 1: Write so let's move on to:

### 1.5.1 1.4 You Code: refactor

Let's strip the logic out of the above code to accomplish the task of the function:

- Send into the function as input a credit card number as a `str`
- Return back from the function as output the issuer of the card as a `str`

To help you out we've written the function stub for you all you need to do is write the function body code.

[ ]:
```
def CardIssuer(card):
    ## TODO write code here they should be the same as lines 3-13 from the code␣
 ↪above
    #card = input("Enter a credit card: ")
```

```
    digit = card[0]
    if digit == '4':
        issuer = "Visa"
    elif digit == '5':
        issuer = "Mastercard"
    elif digit == '6':
        issuer = "Discover"
    elif digit == '3':
        issuer = "American Express"
    else:
        issuer = "Invalid"
    #print(issuer)
    # the last line in the function should return the output
    return issuer
```

[ ]:
```
x = CardIssuer("4873495768923465")
print(x)
```

### 1.5.2 Step 3: Test

You wrote the function, but how do you know it works? The short answer is unless you write code to *test your function* you're simply guessing!

Testing our function is as simple as calling the function with input values where WE KNOW WHAT TO EXPECT from the output. We then compare that to the ACTUAL value from the called function. If they are the same, then we know the function is working as expected!

Here are some examples:

```
WHEN card='40123456789' We EXPECT CardIssuer(card) to return Visa
WHEN card='50123456789' We EXPECT CardIssuer(card) to return MasterCard
WHEN card='60123456789' We EXPECT CardIssuer(card) to return Discover
WHEN card='30123456789' We EXPECT CardIssuer(card) to return American Express
WHEN card='90123456789' We EXPECT CardIssuer(card) to return Invalid Card
```

### 1.5.3 1.5 You Code: Tests

Write the tests based on the examples:

[ ]:
```
# Testing the CardIssuer() function
print("WHEN card='40123456789' We EXPECT CardIssuer(card) = Visa ACTUAL",␣
 ↪CardIssuer("40123456789"))
print("WHEN card='50123456789' We EXPECT CardIssuer(card) = MasterCard ACTUAL",␣
 ↪CardIssuer("50123456789"))

## TODO: You write the remaining 3 tests, you can copy the lines and edit the␣
 ↪values accordingly
```

### 1.5.4 Step 4: Rewrite

The final step is to re-write the original program, but use the function instead. The algorithm becomes

```
input credit card number into variable card
call the CardIssuer function with card as input, issuer as output
print issuer
```

### 1.5.5 1.6 You Code

Re-write the program. It should be 3 lines of code:

- input card
- call issuer function
- print issuer

[ ]:
```
# TODO Re-write the program here, calling our function.
```

## 1.6 Functions are abstractions. Abstractions are good.

Step on the accellerator and the car goes. How does it work? Who cares, it's an abstraction! Functions are the same way. Don't believe me. Consider the Luhn Check Algorithm: https://en.wikipedia.org/wiki/Luhn_algorithm

This nifty little algorithm is used to verify that a sequence of digits is possibly a credit card number (as opposed to just a sequence of numbers). It uses a verfication approach called a **checksum** to as it uses a formula to figure out the validity.

Here's the function which given a card will let you know if it passes the Luhn check:

[ ]:
```
# Todo: execute this code

def checkLuhn(card):
    ''' This Luhn algorithm was adopted from the pseudocode here: https://en.
 ↪wikipedia.org/wiki/Luhn_algorithm'''
    total = 0
    length = len(card)
    parity = length % 2
    for i in range(length):
        digit = int(card[i])
        if i%2 == parity:
            digit = digit * 2
            if digit > 9:
                digit = digit -9
        total = total + digit
    return total % 10 == 0
```

[ ]:
```
checkLuhn('4916945264045429')
```

#### 1.6.1 Is that a credit card number or the ramblings of a madman?

In order to test the `checkLuhn()` function you need some credit card numbers. (Don't look at me... you ain't gettin' mine!!!!) Not to worry, the internet has you covered. The website: http://www.getcreditcardnumbers.com/ is not some mysterious site on the dark web. It's a site for generating "test" credit card numbers. You can't buy anything with these numbers, but they will pass the Luhn test.

Grab a couple of numbers and test the Luhn function as we did with the `CardIssuer()` function. Write at least to tests like these ones:

```
WHEN card='5443713204330437' We EXPECT checkLuhn(card) to return True
WHEN card='5111111111111111' We EXPECT checkLuhn(card) to return False
```

[ ]:
```
#TODO Write your two tests here
print("when card = 5443713204330437, Expec checkLuhn(card) = True, Actual=",␣
 ↪checkLuhn('5443713204330437'))
print("when card = 5111111111111111, Expec checkLuhn(card) = False, Actual=",␣
 ↪checkLuhn('5111111111111111'))
```

### 1.7 Putting it all together

Here's the Algorithm:

```
print the title "credit card validator"
write an interact function with card as input
    get the card issuer
    if the card passes lhun check
        use thumbs up emoji
    else
        use thumbs down emoji
    print in normal text the emoji icon and the card issuer
```

#### 1.7.1 1.7 You Code

[ ]:
```
## TODO Write code here
```

## 2 Metacognition

#### 2.0.1 Rate your comfort level with this week's material so far.

**1** ==> I don't understand this at all yet and need extra help. If you choose this please try to articulate that which you do not understand to the best of your ability in the questions and comments section below.
**2** ==> I can do this with help or guidance from other people or resources. If you choose this level, please indicate HOW this person helped you in the questions and comments section below.
**3** ==> I can do this on my own without any help.
**4** ==> I can do this on my own and can explain/teach how to do it to others.

--== Double-Click Here then Enter a Number 1 through 4 Below This Line ==--

[ ]: