1 TestCases: How this tutoring actually worked for us :)

How to make one?

There are no set design or inbuilt system for building test cases. Since the functions can have much variety, it is hard to create a singular class of Test Cases. Let me give you an example question here.

```
def factorial(n):
    """Returns n!
    >>> factorial(0)
    1
    >>> factorial(3)
    6
    >>> factorial(5)
    120
    """
    "***YOUR CODE HERE***"
```

So how do we know if we could test all possible cases? Well in the case of factorial, we know that n would always have to be a whole number, that is including zero and positive numbers. We also know that factorial is testing either recursion or iteration. So as long as you've tested the base case and 1 or 2 other cases, you would be fine with it.

Let's say this factorial function was in your codingpractice.py. Let's say that you would like to create an autograder on another python file called codingauto.py. Firstly, you must import the functions that one has defined in the codingpractice.py. We do this by the following line.

```
from codingpractice import *
```

In order to import it properly without defining any directories, you must place these two files under the same directory.

Let's try defining a function that would input the expected value, a function, and inputs, and prints whether one gets it correctly or not.

```
def testcase(f, expected, arg):
    """Here are some examples:
    >>> testcase(factorial, 1, 0)
    factorial(0)
    1

    Correct!
    >>> testcase(incorrect_factorial, 120, 5)
    incorrect_factorial(5)
    20

    Incorrect!
    Expected Value: 120
    Actual Value: 20
    """
    "***YOUR CODE HERE***"
```

How to catch printed values?

Receiving returned value of a function is relatively easy. You just call a function and assign it to some variable. The hard part comes when you have to catch the printed values. Back in our first lecture, I've mentioned that printed values do not have actual values within. They are just mere prints that has no actual definition.

So How do we catch it? We will use two inbuilt packages within python: sys, io.

```
import sys, io
```

Using these two packages, you can temporarily set the system's output to a variable that would be a StringIO object. StringIO objects can handle printed outputs and interpret it as strings. Here is how you do it.

```
expected, actual = io.StringIO(), io.StringIO()
sys.stdout = expected
f(*args)
sys.stdout = actual
[print(i) for i in answers]
sys.stdout = sys.__stdout__
a, b = expected.getvalue(), actual.getvalue()
```

And then you will be comparing a and b to see if the printed values match the actual answers. It is important that you must set your sys.stdout back to sys.__stdout__ to ensure that the system's output has been restored to its original state.

Note: This design of catching the printed value is solely designed by Hyun Jae. I believe there are better methods online to catch printed values.

Try creating an autograder for hailstone:

```
def testcase(f, answers, arg):
    """ Here is one example:
    >>> testcase(hailstone, [10, 5, 16, 8, 4, 2, 1], 10)
    hailstone(10)
    10
    5
    16
    8
    4
    2
    1

    Correct!
    """
    "***YOUR CODE HERE***"
```

Multiple Inputs

It is true that certain functions could have more than 1 parameters. We do not want to create a new function every single time a function has different numbers of parameters. This is where we use the syntax '*args'. '*args' represent that it can take as many arguments. However, it is important that this *args must stay as the last parameter. Here is an example on how to use args:

```
def printallargs(*args):
    """ Prints each arguments one by one
    >>> printallargs(1, 2, 3):
    1
    2
    3
    >>> printallargs(0)
    0
    >>> printallargs()
    """
    for i in [*args]:
        print(i)
```

Now, I want you to create a general autograder that could be flexible with the number of arguments inside.

```
def autograderV1(f, expected, *args):
    """ Here are some examples
    >>> def two_add(a, b): return a + b
    >>> def three_add(a, b, c): return a + b - c
    >>> autograderV1(two_add, 5, 2, 3)
    two_add(2, 3)
    5

    Correct!
    >>> autograderV1(three_add, 6, 1, 2, 3)
    three_add(1, 2, 3)
    0

    Incorrect!
    Expected Value: 6
    Actual Value: 0
    """
    "***YOUR CODE HERE***"
```

How to raise Error?

As you can see, in my autograder.pyc, if you get a question wrong, the autograder does not proceed, instead it halts and raises an error. Raising an error is relatively easy:

```
raise ValueError("message")
```

Once the error is raised, the program immediately stops, and the code does not proceed. It also tracebacks to the lines that you were getting this error from.

try and except

One other interesting feature in python is try and except syntax. These syntax catches a certain error and perform a different procedure. Let me show you an example.

```
>>> def f():
...     raise ValueError
...
>>> try:
...     f()
... except ValueError:
...     print("You've gotten a ValueError")
...
You've gotten a ValueError
```

You may have certain cases where you need to perform some procedure when you are getting an error from another method. It is definitely a useful tool in python.

How to make wwpd? (input, eval, break)

There are more stuff you need to know in making wwpd. However, the most important portion is the user input. How to make a user input values, so you could use it to determine whether the user got it correct or not. Here is what I've created.

```
def general_test(n, q):
    print("Question "+str(n))
    while True: #Infinite Loop, so that the user could try it multiple times.
        print(">>> "+q)
        answer = eval(q) #q is a String. eval("2 + 3") -> 5
        i = input()
        if i == "exit":
            raise ValueError("Exiting the test.") #Just for convenience
        if i != str(answer):
            print("Incorrect")
        else:
            print("Correct")
            break #the break literally breaks this while True loop.
    print("-------------")
```

If you run general_test(1, "2 + 3"), question 1 will be presented asking you for the answer for 2 + 3.

Design one for yourself!

This time, I would like you to create a class Testcase that has methods for both checking the return values and also checking the printed values as well! I will give you some examples:

```
import sys, io
class Testcase:
    """ Here are some examples
    >>> testFactorial = Testcase(factorial)
    >>> testFactorial.returntest(120, 5)
    factorial(5)
    120

    Correct!
    >>> testHailstone = Testcase(hailstone)
    >>> testHailstone.printtest([8, 4, 2, 1], 8)
    hailstone(10)
    8
    4
    2
    1

    Correct!
    """
    "***YOUR CODE HERE***"
```

2 Conclusion

Topics we've covered

- Primitives: Integers, Booleans, Strings

- Conditional Statements: if/else

- While Loops

- Functions: def, function vs function call

- Environmental Diagrams: Frames, Primitives vs Objects

- Recursion: Base case, Recursive Step

- Tree Recursion: Recursive Step in a form of a tree

- Lists: List Comprehension, For Loops

- Trees: label, branches, is_leaf

- Object Oriented Programming: __init__, attributes, methods, self

- Linked Lists: first, rest, box and pointers

- Generators: yield, iterable/iterator

- TestCases: sys, io, input, break, eval, *args, try, except

- Numpy: importing packages, googling documentations

Congratulations on finishing Basic Python and Data Structures in 8 weeks!