Hour 1

Recursion:

- Traditional `for` loop:

```
# runs through each char in the password
for i in range(len(password)):

    # for each char, compares to each item in the special chars list
    for j in range(len(SPECIAL_CHARACTERS)):
        if SPECIAL_CHARACTERS[j] == password[i]:
            check_is_special = True
```

- Factorial:
  `n! = n x (n-1) x (n-2)…. x 1`
- Factorial using for loop:

```
def factorial(n):

    total = 1
    for i in range(n, 0, -1):
        total = total * i
    return total
```

- Factorial can also be written as:
  `5! = 5 x 4 x 3 x 2 x 1`
  `5! = 5 x 4!`
  `n! = n x (n - 1)!`
- Instead of writing `!`Factorial, lets say we have a function called `factorial()` that calculates the factorial of a passed in parameter.

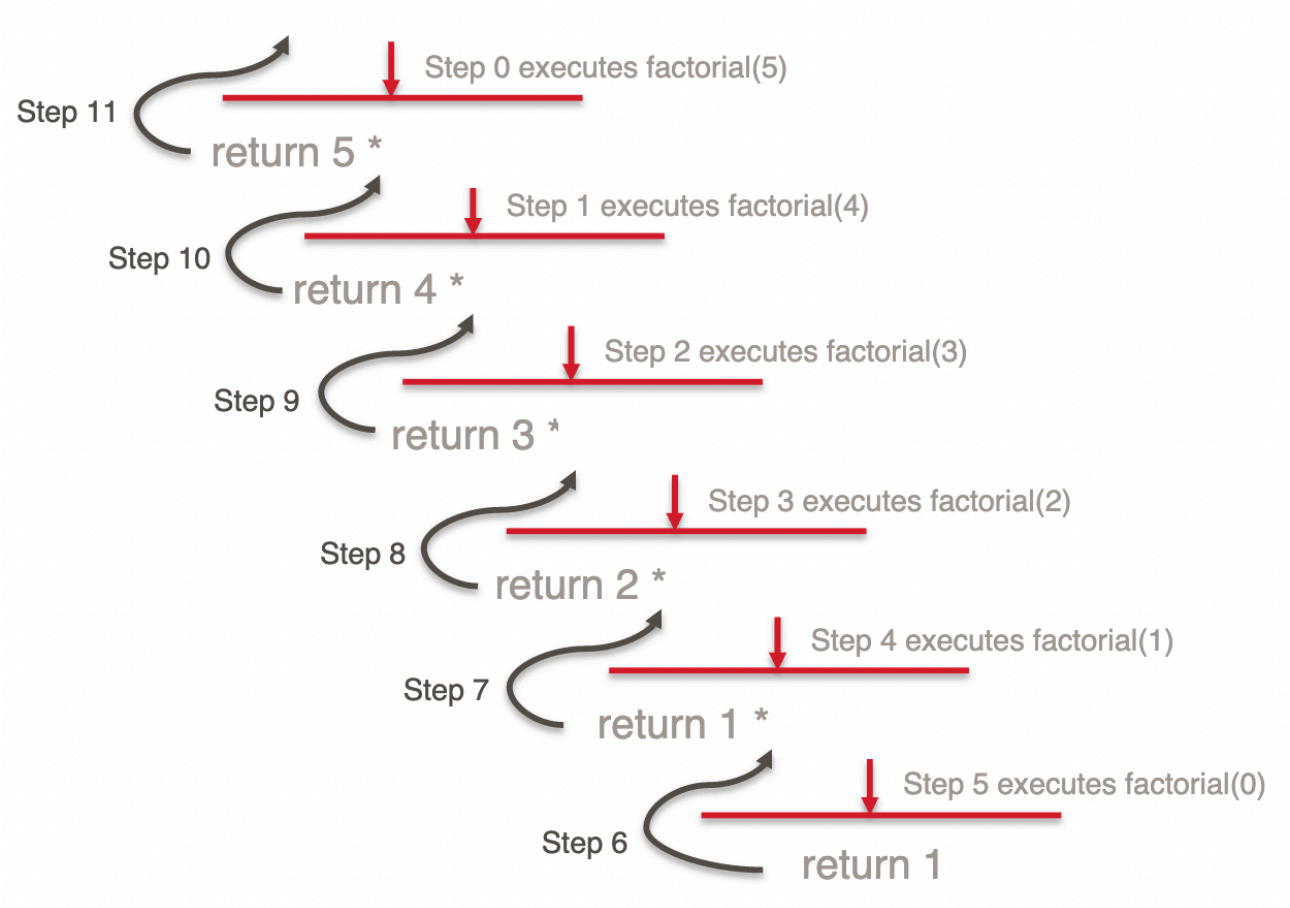
  `factorial(n) = n x factorial(n - 1)`
- Recursion is way to solve problems by **recursively** reducing the problem into smaller bits till we reach the end of problem.

```
def factorial(n):

    return n * factorial(n-1)
```

- But here we haven't specified the base case which is n = 1. This means it would just go into an infinite loop and give you a maximum recursion depth error.
- So completed code would be:

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)
```

- It works by going all the way till the base case and then returning the values back up till it reaches the value of n:



Steps to write a problem using recursion:

- Find a way to break problem into smaller steps
- Establish a base case
- Determine how to return the answer

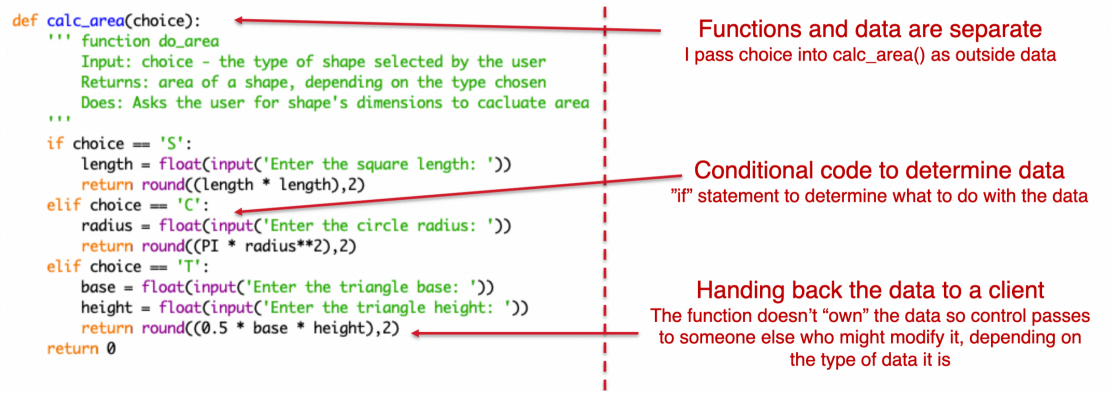
Hour 2

Recursive Palindrome: Exercise:

```
def palindrome(word):

    if not word or len(word) == 1:
        return True

    return palindrome(word[1:-1]) if word[0] == word[-1] else False

def main():
    print(palindrome("aaaaaa"))
    print(palindrome("aaaaa"))
    print(palindrome(""))
    print(palindrome("a"))
    print(palindrome("aaxhjfasa"))

if __name__ == "__main__":
    main()
```

Classes and Objects:

Procedural Approach:



For example: square object -

It is a object that knows its own length and its name.

```
class Square(Shape):
    def __init__(self, length):
        self.length = length
        self.name = 'Square'
```

- Class methods are basically functions inside a class and they can be accessed by a dot .
- Some methods are special with the double underscores before and after.

```
__str__(self)
__eq__(self, other)
__init__(self)
```

- Classes are like blueprints from with objects are built
- A class is a tweet template and every tweet is an object of that class
- Minion Example:

```
class Minion:
    def __init__(self, name):
        self.name = name
        self.color = "yellow"
    def yell(self, word):
        print(word, "!!!!")
    def obey(self):
        do_grus_thing()
```

Creating objects from minion class:

```
m1 = Minion("Kevin")
m1.yell("Banana!")

m2 = Minion("Bob")

the_boss = Gru()
the_boss.give_orders(m1, m2)

m1.obey()
m2.obey()

the_boss.take_over_the_world("WINNING")
```

Self keyword:

- Always refers to the instance of the class
- It changes based on what we are referring to.

Hour 3

Circle Class Example:

```
"""
    Classes Intro
"""

PI = 3.1415

class Circle:

    # Constructor
    def __init__(self, radius):

        #Define its constructor values
        self.radius = radius
        self.color = "blue"

    # Derived Getter
    def get_area(self):
        return PI * self.radius ** 2

    # Basic Getter
    def get_radius(self):
        return self.radius

    def __str__(self):
        return "Circle with radius " + str(self.radius)
```

Driver for Circle:

- This should be put in a separate file and each class should have a separate file of its own.

```
from circle import Circle

def main():

    circle = Circle(2)
    circle2 = Circle(5)

    print(circle.get_area())
    print(circle2)


if __name__ == "__main__":
    main()
```

Hour 3. 5

Testing circle class with unit tests:

```
import unittest
from circle import Circle

PI = 3.1415

class TestCircle(unittest.TestCase):

    # Positive Tests
    def test_init(self):
        c = Circle(2)
        self.assertEqual(c.get_radius(), 2)

    def test_get_area(self):
        c = Circle(2)
        self.assertEqual(c.get_area(), PI * 4)

    # Negative Tests: We expect an error
    def test_bad_init(self):
        with self.assertRaises(ValueError):
            c = Circle(-2)

def main():
    unittest.main(verbosity=3)

main()
```

Square Class Practice:

```
class Square():
    def __init__(self, width):
        self.width = width

    def get_area(self):
        return self.width * self.width

    def get_perimeter(self):
        return self.width * 4

    def set_width(self, width):
        self.width = width
```

```
from square import Square

def main():
    square = Square(5)

    print(square.get_area())
    print(square.get_perimeter())

    square.set_width(10)

    print(square.get_area())
    print(square.get_perimeter())

    print(square)


if __name__ == "__main__":
    main()
```