
PYTHON ASSIGNMENT #3

Quick Review of Assignment #2

The `def` keyword begins a function definition.

"reciprocal" is the name of our function. We can use this function later by calling its name. Function names **MUST** be one word!

These brackets contain the arguments that are passed to your function.

The colon tells python that the rest of the code following is the definition of the function.

```
def reciprocal(x):  
    return 1.0/x
```

The optional "return" keyword allows you to tell python what your function should evaluate to when you call it. In this case, `reciprocal(x)` evaluates to `1.0/x`.

Notice how the function definition is indented? This is important! Python knows that all of the indented code is part of the function definition. Once you are done defining your function, stop indenting. Python figures out the rest. Although IDLE will auto-indent for you, if you ever do it yourself make sure you only use SPACES, not TABS. If you mix spaces and tabs, your program will FAIL horribly and you won't know why.

Note that this function will not do anything unless you specifically call it. To call our example function (in other words, to make it run), you just use its name and supply it with arguments like this: `reciprocal(2)` or `reciprocal(-2)`

IF STATEMENTS

Programming starts to become exciting when your programs can make decisions. Here's how!

Say you wanted to make a program that knows a secret, but will only print the secret if you enter the correct password. Here's what it would look like:

```
PASSWORD = "power overwhelming"  
print = "If you know the password, I will tell the meaning of life."  
answer = raw_input("What's the password? ")  
  
if answer == PASSWORD:  
    print "Ahh, you are the chosen one ..."  
    print "The meaning of life is 42"  
  
print "Okay?"
```

Let's break it down a bit:

The "if" keyword begins the branching of your code.

The next part is called the **condition**. The condition must evaluate to a **BOOLEAN** value: either True or False

The colon tells Python that you're done your condition

```
if answer == PASSWORD :  
    print "Ahh, you are the chosen one ..."  
    print "The meaning of life is 42"  
  
print "Okay?"
```

The indented code will run if your condition is true, otherwise it won't run at all.

This bit of code will run no matter what

Conditional Operators: Here are the most commonly used operators in a conditional statement. They all return either True or False

Operator	Description
A == B	checks to see if A is equal to B
A < B	checks to see if A is less than B
A > B	checks to see if A is greater than B
A != B	checks to see if A is not equal to B
A in B	checks to see if A is a subset of B

Logical Operators: You can also combine multiple conditions with the logical operators AND, OR, and NOT.

x and y If both conditions are true, "and" returns true. If either is false, "and" returns false.
x or y If either condition is true, "or" returns true. If both are false, "or" returns false.
not(x) reverses the truth value of a statement.

Multiple Branches: else and elif

Maybe you have more than 1 branch you want to break your program into. For example, what if you wanted the program to respond to other possible answers? You can use the else and elif keywords:

```
if answer == PASSWORD:  
    print "Ahh, you are the chosen one ..."  
    print "The meaning of life is 42"  
elif answer == "screw you":  
    print "Tsk, Tsk!"  
    print "Access Denied"  
elif answer == "please?":  
    print "Nice try but..."  
    print "Access Denied"  
else:  
    print "Access Denied"
```

The elif stands for "else if" and it lets your program respond differently to different conditions. You can have as many as you want, and they are evaluated in order from top to bottom.

The else is a catch-all at the end that will be the default response if no other condition is True.

EXERCISES

Exercise 1: secret.py

Write a program that keeps a secret (whatever you want) unless the correct password "haxxors" is given. This password isn't supposed to be case sensitive, so let the user see the secret if they enter "haxxors" or "HAXXORS" or "Haxxors". Also, make your program respond to 3 other possible answers the user might give. See the reference solution for an example and try to make yours as close to the reference solution as possible. 5 marks.

Exercise 2: gradebot.py

Write a program that will help the user calculate their grade and percent on a test.

1. Ask the user for their test mark
2. Ask the user what the test was out of.
3. Calculate and print their percentage grade.
4. Figure out what letter grade that mark is, and write a message to the user to congratulate them on a job well done or encourage them to do better next time.
5. Find a way to prevent the user from crashing the program if they make the test out of zero, and deal with negative percentages and percentages greater than 100.

Percentage Lower Bounds	Letter Grade
86	A
73	B
67	C+
60	C
50	C-
0	F

10 marks altogether. See the reference solution for an example.

Exercise 3: magic8ball.py

First, import the `randint()` function from the module called `random`. When you call `randint()`, you pass it two numbers, and `randint` will chose a random integer between those two numbers. For example, `randint(0, 3)` will return either 0, 1, 2, or 3. `randint(4, 7)` will randomly return either 4, 5, 6, or 7.

Now, use `randint()` to simulate a [Magic 8 Ball](#). The user should be able to ask a yes or no question, and the Magic 8 Ball will return a random vague answer. You should have at least 10 possible vague answers that can be returned. See the reference solution for an example of this program in action (which uses the 20 official responses as per Wikipedia). Worth 5 marks

Exercise 4: rps.py

Write a program that simulates a game of [rock, paper, scissors](#) between a human user and the computer.

1. Use `raw_input()` to get the human player's choice.
2. Use `randint()` and if statements to get the player's choice. Print it so we can see what it chose.
3. Use if statements to evaluate who is the winner and who is the loser. There can also be a tie! Insult the user accordingly.
4. If the user enters an incorrect choice (ie: laser sword), then don't let them play. You can import the `exit()` function from the `sys` module to exit the program early if you need to.
5. Include everything you see in the reference solution, although you can insert your own funny remarks.

See the reference solution for an example. Worth 10 marks!