**Introduction to Python Programming**



**Hello, World!**

This is a comment that is not run by the program

Write a program that prints out the message:

Hello World!

# Enter your code for "Hello, World!" here.

print("Hello, World!")​

**Hello to you, inparticular.**

Write a program that asks the user's name, then greets them by saying hello. For example, my interaction with your program should look like:

What is your name? Vic

Hello, Vic

# Enter your code for "Hello to you, too" here.

name = input('What is your name? ')

Input is put in a variable called “name”

print ('Hello, ' + name)

**Doing maths**

Python can do all the basic maths you would expect. The main operators are:

| **operator** | **operation** |
| --- | --- |
| + | addition |
| - | subtraction |
| \* | multiplication |
| / | division |
| \*\* | to the power of |

The *power* operator is represented by a double asterisk (\*\*) and works like this:

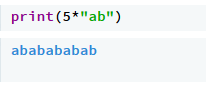
n = 10

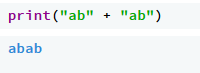
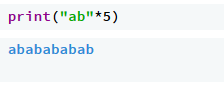
print(n\*\*2)

“100”

To write encryptions and decryptions we need to…

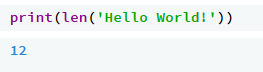
# Do maths with words!

Something you might not expect is that some of the maths operators in Python also work on strings. You've already seen that we can do string addition:

# Investigating strings

Once we've stored a word or sentence in a string variable, we might want to investigate some of its properties. The simplest of these it its length.

We can find out how long a string is using the len function: 

Calculates the number of characters in “hello world”

Try this code:

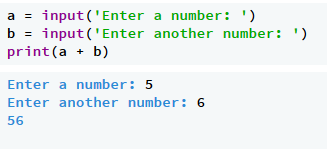
# Enter your code for "Length of a string" here.

string = input('Enter text: ')

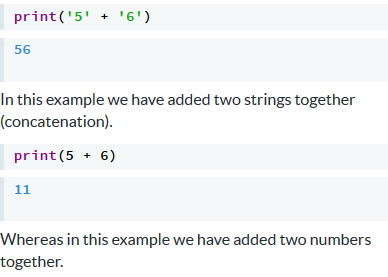
print (len(string))

# Numbers and strings are different

Let's ask the user for two numbers and then add them together:

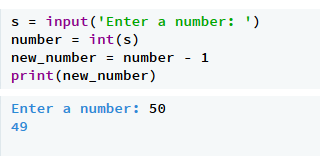
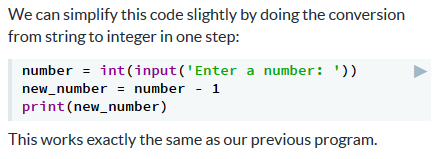


Crazy! It doesn't do what we want at all - the answer should be 11. The reason is that numbers and strings are treated differently in Python because they are different types of information. The problem here is that input always gives us back a string regardless of what the user enters. Let's see that again without the input:



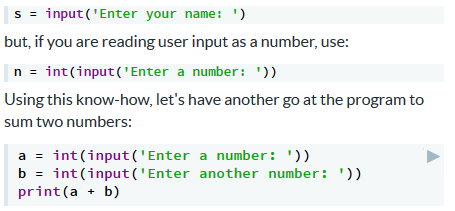
# Converting strings to numbers

If you want to use a string as a number you must first convert it to one. You can convert a string to an integer (a whole number) with the int function. Look at the following code which reads in a number and then prints out the number one less than it:



# Reading in numbers

We will talk a lot more about type conversions later in the course. For now, if you want to read user input as a string, use:



# 1-Up

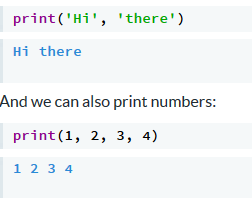
Write a program that reads in a number (integer) as input, adds 1 to it, and then prints out the result. Here is an example interaction between your program and the user:

Number = int(input('Please enter a number: '))

Number = Number + 1

print (Number)

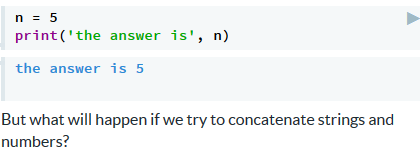
# Printing out numbers and strings

You have already seen the print statement many times. It writes values to the terminal and then writes a new line (that is, shifts the cursor to the beginning of the next line).

Remember we can print letters and words:

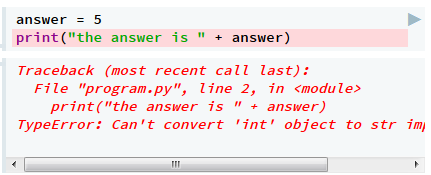
In both these cases we are passing in multiple arguments to the print function, by using a comma ( , ) to separate them. When you do this, a space is automatically inserted between each expression.

When you are passing multiple arguments to the print function (using a comma to separate them), you don't need to convert the numbers to strings:



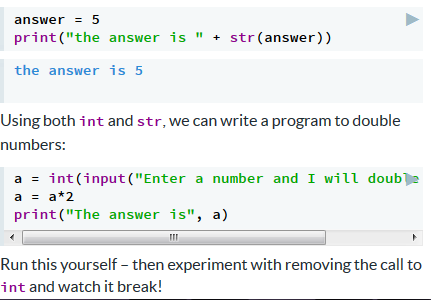
# Converting numbers to strings

If you don't want a space between the printed expressions, you must concatenate strings yourself. However, if we try this we get an error:



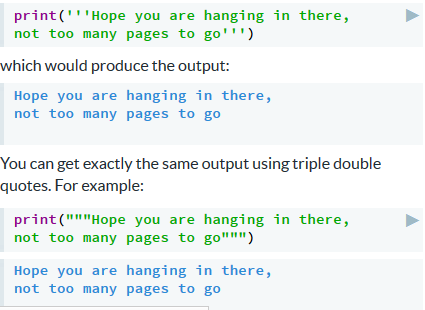
The + operator is confused here because we cannot add a string and an integer. When this happens, Python gives a runtime error (in this case, a TypeError). The program syntax is correct, but the program is logically broken.

To fix it we need to use the str function to turn the integer back to a string:



# Printing on multiple lines

Sometimes you'll need to print out longer messages which will extend over multiple lines. To do this you can use strings with triple single or double quotation marks (''' or """). For example:



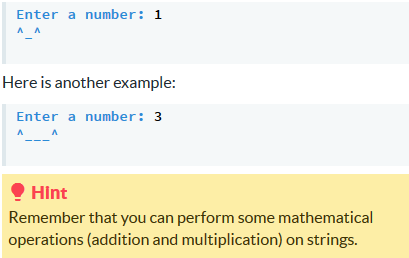
# Python makes me ^\_^

After starting to learn about Python, you want to express you newfound joy to your friends. Your favourite happy emoticon is the [Japanese-style emoticon](http://en.wikipedia.org/wiki/Japanese_Emoticons#Japanese_style) ^\_^.

When you finish solving a question, you're super happy and want to express a larger amount of happiness by adding more underscores between the carets: ^\_\_\_\_\_^.

After telling a couple of your friends how happy you are, writing out these emoticons is becoming tiring. You decide to use your new programming skills to help you out!

Write a Python program to read in a number that indicates how happy your emoticon should be. For example:



This code will read in a number and use the number to make the emoticon smile.

# Your code for "Python makes me ^\_^" here.

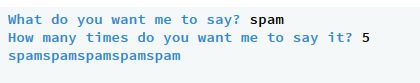
number = int(input('Enter a number: '))

print('^' + '\_'\*number + '^')

# Repeaterbot

In your travels through space time you come across a robot with the useful (?) ability to repeat everything you say, as many times as you want. Write a program to control this robot.

Your program should work like this:



# Code for "Repeaterbot"

word = input('What do you want me to say? ')

number = int(input('How many times do you want me to say it? '))

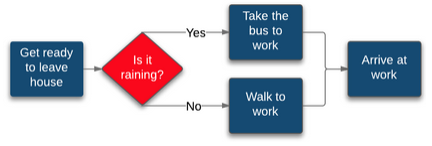
print (word\*number)

# Decisions, decisions, decisions!

# Why do we need decisions?

So far the programs we have written are quite simple, when run they execute a list of statements from top to bottom. Because of this they will run in exactly the same way each time which is very useful for solving some tasks but not others.

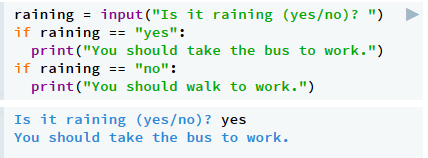
In the real world we often need to make decisions based on a situation. For example, how we are going to travel to work or school today:



The diagram shows the two different options of how to get to work, either by taking the bus or walking. The red diamond which asks 'is it raining?' determines which way the program will flow.

We can implement this in Python using control structures. The first control structure we will look at is the if statement.

# What if it is raining?

Let's write a program that would help us make a decision about how to get to work. Of course the computer can't check the weather on its own, so it will have to ask us.

Try running the programming and seeing what happens when you answer yes and no.

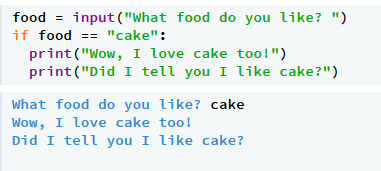
The first if statement controls the execution of the first print statement because it is indented under it. If the user says it is raining, the program advises catching the bus.

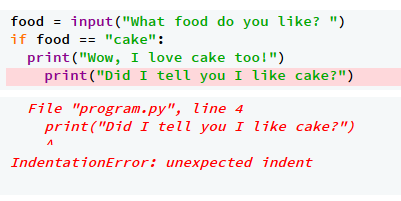
The second if statement controls the execution of the second print statement because it is indented under it. If the user says it is raining, the program advises walking.

The if statement is controlling the program by changing its behaviour depending on the user's input. **This why it is called a control structure!**

# Controlling a block of code

if statements (and other control structures) control a chunk of code called a block. In Python, a statement is indented to indicate it belongs to a block. For example:

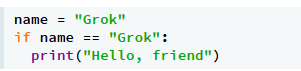
The indentation must be to the same depth for every statement in the block and must be more than the statement controlling the block. This example is broken because the indentation of the two lines is different.

You can fix it by making both print statements indented by the same number of spaces (usually 2 or 4).

The block is often called the body of the control structure. Every control structure in Python has a colon followed by an indented block.

# Assignment vs comparison

You will notice in our examples that we are using two equals signs to check whether the variable is equal to a particular value:



The equals symbol = means assignment.

== means EQUALS EXACTLY THE SAME

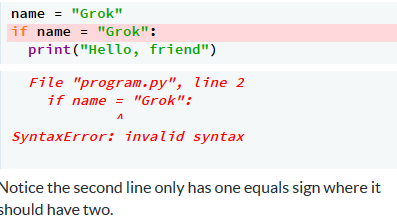
​

This can be very confusing for beginner programmers.

A single = is used for assignment. This is what we do to set variables. The first line of the program above is setting the variable name to the value "Grok" using a single equals sign.

A double == is used for comparison. This is what we do to check whether two things are equal. The second line of the program above is checking whether the variable name is equal to "Grok" using a double equals sign.

If you do accidentally mix these up, Python will help by giving you a SyntaxError. For example, try running this program:



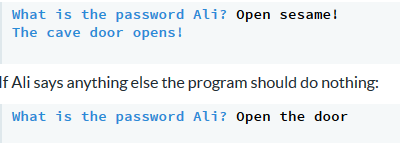
**Now it’s up to YOU!**

# Open sesame!

Ali Baba is trying to open a cave door to find the treasure. His brother overhears thieves saying the password, but he can't quite work out what it is.

Write a program to ask Ali for a password. If the password is correct, the cave door will be opened. Otherwise nothing will happen.

Your program should work like this when Ali says "Open sesame!":

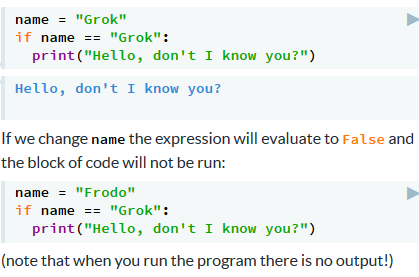


**Now it’s up to YOU!**

# True or False?

if statements allow you to make yes or no decisions. In Python (and most programming languages) these are called True and False.

When you use an if statement Python evaluates a conditional expression to determine if it is True or False. If the expression evaluates to True then the block controlled by the if statement will be run:

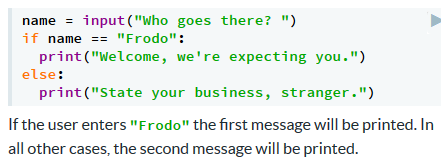


You can check whether the conditional expression is evaluating to True or False by testing it directly:



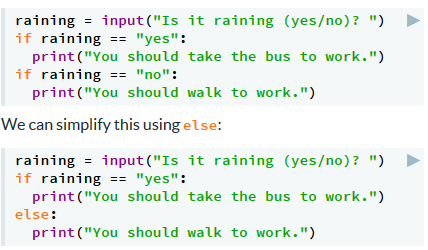
# Decisions with two options

All of the examples so far have done something when a condition is True, but nothing when the condition is False. In the real world we often want to do one thing when a condition is True and do something different when a condition is False.

For example, we may want to welcome a friend, but find out more about someone we don't know. We can do that using an else clause:

# If it isn't raining...

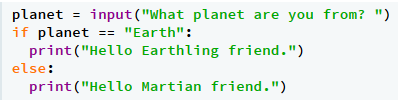
Using the else clause can make code neater and easier to read. For example, let's go back to our raining example from the first slide:



Here, either the first or second print statement will be executed but not both. Notice the condition in the else clause must be followed by a : character, just like the if statement.

As well as being neater, the else statement is more efficient because it does not have to perform another conditional evaluation.

# When options are limited

The if/else statement is good for making decisions when there are only two options (friend or foe, true or false, wet or dry). When there are more than two possibilities, the results can be confusing. This is something to watch out for.

For example, look at this program that tries to work out whether the user is an Earthling or a Martian:

If the user enters "Earth" the response makes sense. However, if the user enters "Jupiter" the response doesn't really make sense (try it).

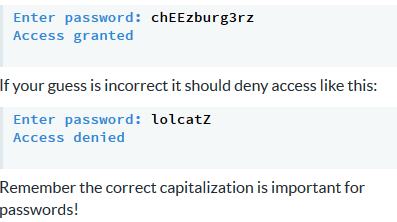
This is because in the real world, there are more than two planets, but our program only deals with two cases. We will solve this problem later in the module.

**Now it’s up to YOU!**

# Access denied

You are trying to log in to your old computer, and can't remember the password. You sit for hours making random guesses... I'm sure you thought it was funny back when you came up with that password (chEEzburg3rz).

Write a program that tells you whether your guess is correct. If it is correct, it should grant access like this:

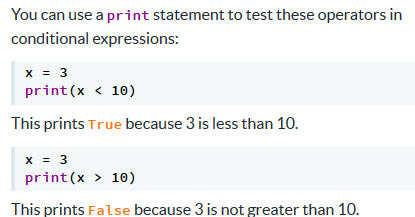


**Now it’s up to YOU!**

**How do we compare things?**

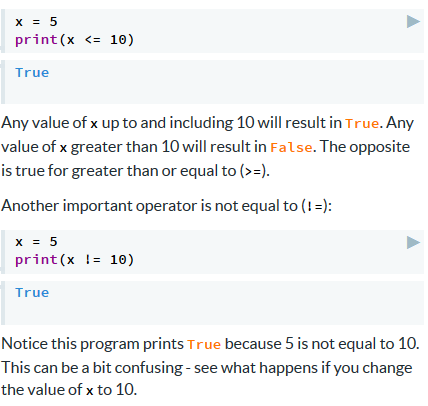
So far we have only checked whether two things are equal. However, there are other ways to compare things (particularly when we are using numbers). We can use the following *comparison operators* in if statements:

| **Operation** | **Operator** | |
| --- | --- | --- |
| equal to | == | |
| not equal to | != |
| less than | < |
| less than or equal to | <= |
| greater than | > |
| greater than or equal to | >= |

You can use a print statement to test these operators in conditional expressions:

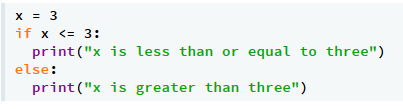
# Experimenting with comparison

Let's try some more examples to demonstrate how conditional operators work. Firstly, we have less than or equal to (<=)



# Making decisions with numbers

Now we can bring together everything we've learned in this section, and write programs that make decisions based on numerical input. The example below makes two decisions based on the value in x:

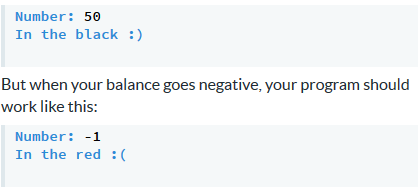


Try assigning different values to x to change how the program executes.

**Now it’s up to YOU!**

# In the black

Write a program that works out whether your bank balance is **in the red**. If the number of dollars is zero or positive it should work like this:

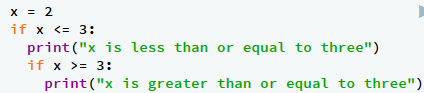


**Now it’s up to YOU!**

# Decisions within decisions

The body of an if statement may contain another if statement. This is called nesting.

In the program below we have indented one if statement inside another one. That means the second condition will only be tested if the first condition is True.



There is only one value of x that will cause both of these messages to be printed. Experiment to work out what it is.

# Decisions with multiple options

If you want to consider each case separately, we need to test more conditions. One way of doing this would be using nesting, as shown below:

You can test as many cases as you want by using multiple elifs.

# Interplanetary visitor

Remember our program that tried to greet interplanetary visitors correctly? We can now extend this so that it makes sense regardless of what planet the user is from.

**Now it’s up to YOU!**

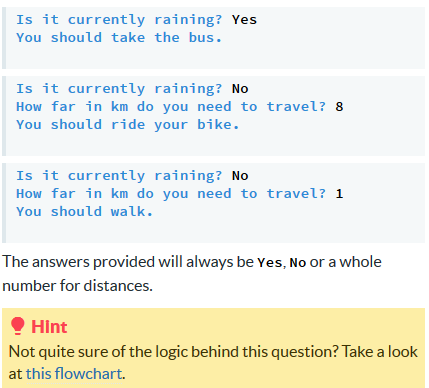
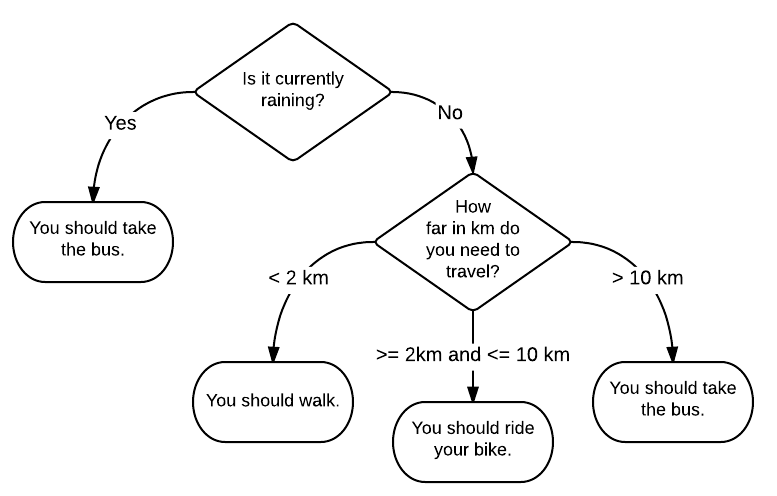
# Method of transportation

Write a program to tell you which method of transport to use for your next outing.

The first step in this decision is based on the weather. If it is currently raining, you should take the bus.

If it is not currently raining, your method of transport should be determined by the distance you need to travel. If the distance is greater than 10km, you should take the bus. If it is between 2km and 10km (inclusive), you should ride your bike, and if it less than 2km, you should walk. The distance should always be a whole number.

Your program should only ask for the distance if it is relevant to the answer. That is, if it is raining, it shouldn't ask you how far you need to travel.



**Now it’s up to YOU!**

# Strings within a string

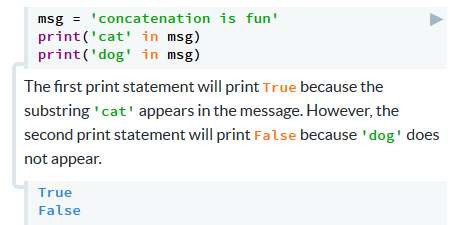
So far we have written a lot of programs that take input from the user in the form of a string (a letter, word or sentence). In this module we are going to learn how to manipulate these strings so that we can write more interesting programs.

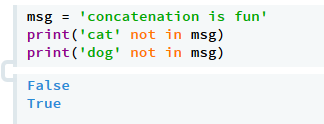
The first thing we are going to do is check the contents of a string, to see if it contains a particular character. To do this, you can use the conditional operator in, like this:



# Substrings within a string

We can also check for groups of characters (or substrings). A substring could represent part of a word, or one word in a phrase. For example:

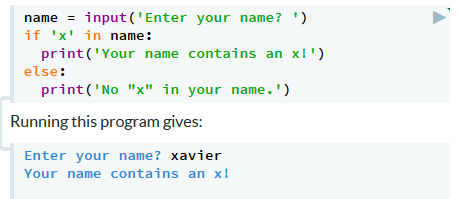


Try changing the message and seeing what happens.

The not in operator does the opposite to in. It returns True when the string does not contain the substring:

# Making decisions with strings

Like the other conditional expressions, we can now use these in an if statement to make decisions. For example to check if a person’s name contains an x character you can write:



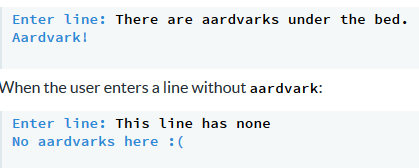
Notice that at the moment the program can only deal with lowercase letters (try entering Xavier). We'll solve this problem in the next section.

**Now it’s up to YOU!**

# Aardvark!

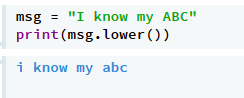
The [aardvark](http://animaldiversity.ummz.umich.edu/site/accounts/information/Orycteropus_afer.html) ("digging foot") is a medium-sized, burrowing, nocturnal mammal native to Africa.

Write a program to detect aardvarks in an input string. Your program needs to find cases where the aardvark appears in lower case (it could be inside a longer word). For example:

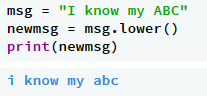


**Now it’s up to YOU!**

# Changing text to lowercase

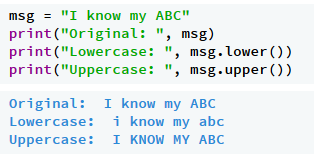
In our previous example we could only compare lowercase text (i.e. not capital letters). However, we often want to change the case from uppercase to lowercase or vice versa. To do this, and many other string manipulations we can use string methods.

A method is a chunk of code that lets you modify or return information about the string. This is best demonstrated with an example.

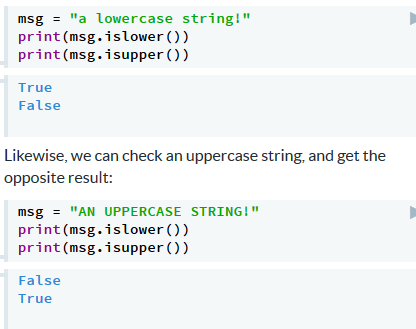
The method we are using here is lower. The msg string contains a message in mixed case, and when you call the lower method it returns a message in lowercase only.

If you want to store the new, lowercase message, you can assign it to a new variable like this:

# Changing text to uppercase

Of course we might want to do the opposite, and change our string to uppercase letters only. The method for doing this is, not surprisingly, called upper. So upper and lower are part of a family of methods that return a modified version of the string. The important thing to note is that the original string is unchanged. So for example, if we print msg it still has a mix of upper and lowercase letters:

# Testing the case of a string

Now we know how to change the case of a string, but how do we know what case something is? We might need to check whether a user has entered upper or lowercase characters (for example, if they were entering a password.)

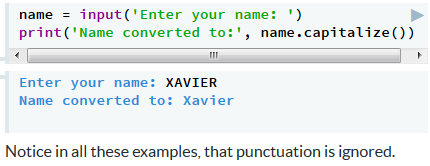
We can do this using the isupper and islower methods. These are similar to conditional operators - they return True or False depending on the input.

Here's an example that checks a lowercase string:

# Making decisions about case

Just like before, we can now use these in an if statement to make decisions. For example to check if a user has given their name in the correct case, you could write:

If we wanted to correct the user's input so that their name had traditional English capitalisation (the first letter capitalised), we could use the capitalize method:

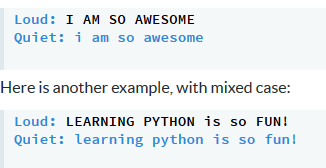


**Now it’s up to YOU!**

# Don't SHOUT!

We all know how annoying it is when people on forums **SHOUT** all the time. Write a program to convert their SHOUTING (uppercase characters) to polite talking (lowercase characters).

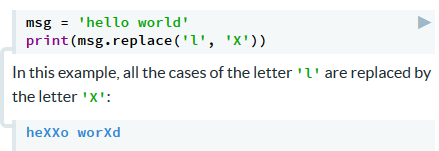
An interaction with your program should look like this:



**Now it’s up to YOU!**

# Replacing parts of a string

You can replace part of a string (a substring) with another substring using the replace method. This requires you to pass the string data you wish to replace and the string you wish to replace it with as arguments to the method.



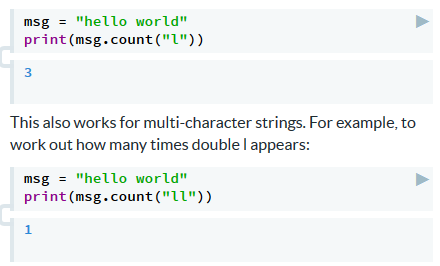
You can replace multiple characters at once:



# Counting the characters in a string

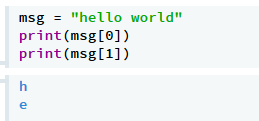
Another useful string method is count, which allows you to count how many times a substring is contained in another string.

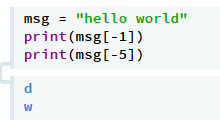
For example, to work out how many times 'l' appears in 'hello world':



Remember that the convention for calling string methods is that the string we are manipulating comes first, and then the method name, with any other information that is required passed in as arguments.

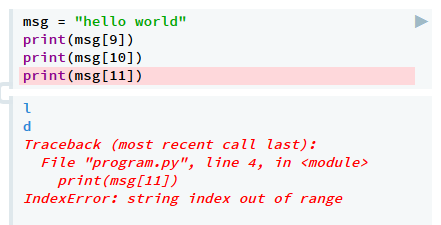
# Getting an individual character

Often we need to access individual characters in a string. Accessing a single character is done using the square bracket subscripting or indexing operation:

You might be wondering why we use 0 to access the first character in the string -- why wouldn't we be using a 1? This is because in programming, and in general in computer science, we start counting from 0 rather than from 1. So the first character in a string is at index 0, the second character is at index 1, the third character is at index 2, and so on.

You can also access strings from the other end using a negative index:

# Characters that don't exist

If you try and access a character past the end of the string, Python will throw an error and your program will crash. Because of this, is important to make sure that you're always trying to access a character in the string which exists. Here is an example of what happens when you try to access a character which is past the end of the string:

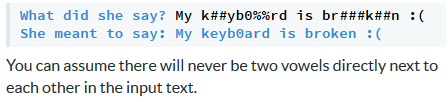
The string 'hello world' only has 11 characters, and we are trying to access character 12 (remember we start counting from 0).

**Now it’s up to YOU!**

# Broken keyboard

Your friend's keyboard is misbehaving, and her "a", "e", and "o" keys are broken. To compensate, when she wants to type an o, she types ###. For an e she types ##, and for an a she types %%. Fed up with trying to interpret this ridiculous code, you decide to write a program to decipher her messages instead.

Write a program to read in some text typed by your friend, and output the corrected text. Your program should work like this:



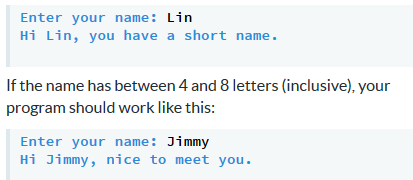
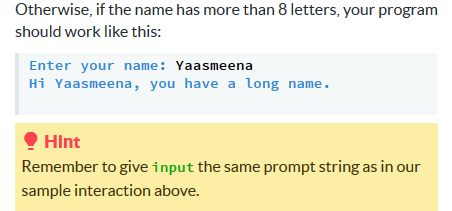
**Now it’s up to YOU!**

**Now it’s up to YOU!**

# Short / long names

Write a program that checks how long a name is. The program should take a name as input from the user.

If the name has 3 or fewer letters, your program should work like this:



**Now it’s up to YOU!**