Strings

When printing things in Python, we are supplying a text block that we want to be printed. Text in Python is considered a specific type of data called a *string*. A string, so named because they're a series of letters, numbers, or symbols connected in order — as if threaded together by string. Strings can be defined in different ways:

```
print "This is a good string"
print 'You can use single quotes or double quotes for a string'
```

Above we printed two things that are strings and then attempted to print two things that are not strings. While double-quotes (") and single-quotes (') are both acceptable ways to define a string, a string needs to be opened and closed by the same type of quote mark.

We can combine multiple strings using `+`, like so:

```
print "This is " + "a good string"
```

This code will print out "This is a good string".

Handling Errors

As we get more familiar with the Python programming language, we run into errors and exceptions. These are complaints that Python makes when it doesn't understand what you want it to do. Everyone runs into these issues, so it is a good habit to read and understand them. Here are some common errors that we might run into when printing strings:

```
print "Mismatched quotes will cause a SyntaxError'
print Without quotes will cause a NameError
```

```
number_of_fish_caught = 10
fish_in_clarks_pond = fish_in_clarks_pond - number_of_fish_caught
```

In the above example, we start with 50 fish in a local pond. After catching 10 fish, we update the number of fish in the pond to be the original number of fish in the pond minus the number of fish caught. At the end of this code block, the variable `fish_in_clarks_pond` is equal to 40.

Updating a variable by adding or subtracting a number to the original contents of the variable has its own shorthand to make it faster and easier to read.

```
money_in_wallet = 40
sandwich_price = 7.50
sales_tax = .08 * sandwich_price

sandwich_price += sales_tax
money_in_wallet -= sandwich_price
```


In the above example, we use the price of a sandwich to calculate sales tax. After calculating the tax we add it to the total price of the sandwich. Finally, we complete the transaction by reducing our `money_in_wallet` by the cost of the sandwich (with tax).

Comments

Most of the time, code should be written in such a way that it is easy to understand on its own. However, if you want to include a piece of information to explain a part of your code, you can use the `#` sign. A line of text preceded by a `#` is called a *comment*. The machine does not run this code — it is only for humans to read. When you look back at your code later, comments may help you figure out what it was intended to do.

Similarly, if we have a string like `"7"` and we want to perform arithmetic operations on it, we must convert it to a numeric datatype. We can do this using `int()`:

```
number1 = "100"
number2 = "10"

string_addition = number1 + number2
#string_addition now has a value of "10010"

int_addition = int(number1) + int(number2)
#int_addition has a value of 110
```

If you use `int()` on a floating point number, it will round the number down. To preserve the decimal, you can use `float()`:

```
string_num = "7.5"
print int(string_num)
print float(string_num)
```

```
>>> 7
>>> 7.5
```


STRINGS & CONSOLE OUTPUT

Strings

Another useful data type is the string. A string can contain letters, numbers, and symbols.

```
name = "Ryan"
age = "19"
food = "cheese"
```

1. **In the above example, we create a variable `name` and set it to the string value `"Ryan"`.**
2. **We also set `age` to `"19"` and `food` to `"cheese"`.**

String methods

Great work! Now that we know how to store strings, let's see how we can change them using string methods.

String methods let you perform specific tasks for strings.

We'll focus on four string methods:

1. `len()`
2. `lower()`
3. `upper()`
4. `str()`

Let's start with len(), which gets the length (the number of characters) of a string!

lower()

Well done!

You can use the `lower()` method to get rid of all the capitalization in your strings.

You call `lower()`like so:

```
"Ryan".lower()
```

which will return `"ryan"`.

upper()

Now your string is 100% lower case! A similar method exists to make a string completely upper case.

str()

Go With the Flow

Just like in real life, sometimes we'd like our code to be able to make decisions.

The Python programs we've written so far have had one-track minds: they can add two numbers or `print` something, but they don't have the ability to pick one of these outcomes over the other.

Control flow gives us this ability to choose among outcomes based on what else is happening in the program.

Compare Closely!

Let's start with the simplest aspect of control flow: comparators. There are six:

Equal to (`==`)

```
>>> 2 == 2
True
>>> 2 == 5
False
```

Not equal to (`!=`)

```
>>> 2 != 5
True
>>> 2 != 2
False
```

Less than (`<`)

```
>>> 2 < 5
True
>>> 5 < 2
False
```

Looking at the example above, in the event that `some_function()` returns `True`, then the indented block of code after it will be executed. In the event that it returns `False`, then the indented block will be skipped.

Also, make sure you notice the colons at the end of the `if` statement. We've added them for you, but they're important.

Else Problems, I Feel Bad for You, Son...

The `else` statement complements the `if`statement. An `if`/`else` pair says: "If this expression is true, run this indented code block; otherwise, run this code after the else statement."

Unlike `if`, `else` doesn't depend on an expression. For example:

```
if 8 > 9:
  print "I don't printed!"
else:
  print "I get printed!"
```


I Got 99 Problems, But a Switch Ain't One

`elif` is short for "else if." It means exactly what it sounds like: "otherwise, if the following expression is true, do this!"

```
if 8 > 9:
  print "I don't get printed!"
elif 8 < 9:
  print "I get printed!"
else:
  print "I also don't get printed!"
```

Nice work! Now that you have it all together, let's take a trip.

What if we went to Los Angeles for 5 days and brought an extra 600 dollars of spending money?

Instructions

1.

After your previous code, `print` out the `trip_cost(` to `"Los Angeles"` for `5`days with an extra `600` dollars of spending money.

Don't forget the closing `)` after passing in the 3 previous values!