**Multi-Line Comments**

The # sign will only comment out a single line. While you could write a multi-line comment, starting each line with #, that can be a pain.

Instead, for multi-line comments, you can include the whole block in a set of triple quotation marks:

**Math**

Great! Now let's do some math. You can add, subtract, multiply, divide numbers like this

addition = 72 + 23

subtraction = 108 - 204

multiplication = 108 \* 0.5

division = 108 / 9

Set the variable count\_to equal to the sum of two big numbers.

a = 1 +9

b = 2-10

c = 6 + 8

count\_to = a+b+c

print count\_to

**Exponentiation**

All that math can be done on a calculator, so why use Python? Because you can combine math with other data types (e.g. **booleans**) and commands to create useful programs. Calculators just stick to numbers.

Now let's work with exponents.

eight = 2 \*\* 3

In the above example, we create a new variable called eight and set it to 8, or the result of 2 to the power to 3 (2^3).

Notice that we use \*\* instead of \* or the multiplication operator.

**Modulo 求余数**

Our final operator is **modulo**. **Modulo**returns the remainder from a division. So, if you type 3 % 2, it will return 1, because 2 goes into 3 evenly once, with 1 left over.

**Instructions**

Use modulo to set spam equal to 1. You can use any two numbers that will leave a remainder of 1 to do this.

**Bringing It All Together**

Nice work! So far, you've learned about:

* **Variables**, which store values for later use
* **Data types**, such as numbers and booleans
* **Whitespace**, which separates statements
* **Comments**, which make your code easier to read
* **Arithmetic operations**, including+, -, \*, /, \*\*, and %

**Instructions**

Let's put our knowledge to work.

1. Write a single-line comment on [line 1](javascript:void(0)). It can be anything! (Make sure it starts with #)
2. Set the variable monty equal toTrue.
3. Set another variable python equal to 1.234.
4. Set a third variable monty\_pythonequal to python squared.

**Strings 说白了就是加双引号**

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 foodto "cheese".

Strings need to be within quotes.

**Instructions**

Create a new variable brian and assign it the string "Hello life!".

**Escaping characters 用\**

There are some characters that cause problems. For example:

'There's a snake in my boot!'

This code breaks because Python thinks the apostrophe in 'There's'ends the string. We can use the backslash to fix the problem, like this:

'There\'s a snake in my boot!'

**Instructions**

Fix the string in the editor!

**Access by Index n指的是第几个字母 ，从0开始**

Great work!

Each character in a string is assigned a number. This number is called the**index**. Check out the diagram in the editor.

c = "cats"[0]

n = "Ryan"[3]

1. In the above example, we create a new variable called c and set it to"c", the character at index zero of the string "cats".
2. Next, we create a new variable called n and set it to "n", the character at index three of the string"Ryan".

In Python, we start counting the index from zero instead of one.

**Instructions**

On [line 13](javascript:void(0)), assign the variablefifth\_letter equal to the fifth letter of the string "MONTY".

Remember that the fifth letter is not at index 5. Start counting your indices from zero.

The string "PYTHON" has six characters,

numbered 0 to 5, as shown below:

+---+---+---+---+---+---+

| P | Y | T | H | O | N |

+---+---+---+---+---+---+

0 1 2 3 4 5

So if you wanted "Y", you could just type

"PYTHON"[1] (always start counting from 0!)

**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!

**Instructions**

1. On [line 1](javascript:void(0)), create a variable namedparrot and set it to the string"Norwegian Blue". On [line 2](javascript:void(0)), typelen(parrot) after the word print, like so: print len(parrot). The output will be the number of letters in "Norwegian Blue"!
2. **lower()**
3. Well done!
4. You can use the lower() method to get rid of all the capitalization in your strings. You call lower() like so:
5. "Ryan".lower()
6. which will return "ryan".
7. **Instructions**
8. Call lower() on parrot (after print) on [line 3](javascript:void(0)) in the editor.

答案：

parrot = "Norwegian Blue"

**print parrot.lower ()**

**upper()**

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

**Instructions**

Call upper() on parrot (after printon [line 3](javascript:void(0))) in order to capitalize all the characters in the string!

**str()**

Now let's look at str(), which is a little less straightforward. The str()method turns non-strings into strings! For example:

str(2)

would turn 2 into "2".

**Instructions**

1. Create a variable pi and set it to3.14 on [line 4](javascript:void(0)).
2. Call str(pi) on [line 5](javascript:void(0)), after print.

**Dot Notation**

Let's take a closer look at why you uselen(string) and str(object), but dot notation (such as "String".upper()) for the rest.

lion = "roar"

len(lion)

lion.upper()

Methods that use dot notation only work with strings.

On the other hand, len() and str()can work on other data types.

**Instructions**

1. On [line 3](javascript:void(0)), call the len() function with the argument ministry.
2. On [line 4](javascript:void(0)), invoke the ministry's.upper() function.
3. **Printing Strings**
4. The area where we've been writing our code is called the **editor**.
5. The **console** (the window in the upper right) is where the results of your code is shown.
6. print simply displays your code in the console.
7. **Instructions**
8. Print "Monty Python" to the console.

**Printing Variables 反是定义了的，print 后不需要””**

Great! Now that we've printed strings, let's print variables

**Instructions**

1. Declare a variable calledthe\_machine\_goes and assign it the string value "Ping!" on [line 5](javascript:void(0)).
2. Go ahead and printthe\_machine\_goes in [line 6](javascript:void(0)).

**[?](https://www.codecademy.com/courses/python-beginner-sRXwR/2/2?curriculum_id=4f89dab3d788890003000096)**

**[Hint](https://www.codecademy.com/courses/python-beginner-sRXwR/2/2?curriculum_id=4f89dab3d788890003000096)**

Make sure you're setting your variable like this:

"""Assign the string "Ping!" to

the variable the\_machine\_goes on

line 5, then print it out on line 6!"""

the\_machine\_goes = "Ping!"

print the\_machine\_goes

**String Concatenation**

You know about strings, and you know about arithmetic operators. Now let's combine the two!

print "Life " + "of " + "Brian"

This will print out the phrase Life of Brian.

The + operator between strings will 'add' them together, one after the other. Notice that there are spaces inside the quotation marks after Lifeand of so that we can make the combined string look like 3 words.

Combining strings together like this is called **concatenation**. Let's try concatenating a few strings together now!

**Instructions**

Let's give it a try. Print the concatenated strings "Spam ", "and ","eggs" on [line 3](javascript:void(0)), just like the example above.

Make sure you include the spaces after "Spam " and "and ".

**Explicit String Conversion**

Sometimes you need to combine a string with something that isn't a string. In order to do that, you have to convert the non-string into a string.

print "I have " + str(2) + " coconuts!"

This will print I have 2 coconuts!.

The str() method converts non-strings into strings. In the above example, you convert the number 2into a string and then you concatenate the strings together just like in the previous exercise.

Now try it yourself!

**Instructions**

1. Run the code as-is. You get an error!
2. Use str() to turn 3.14 into a string. Then run the code again.
3. **String Formatting with %, Part 1**
4. When you want to print a variable with a string, there is a better method than concatenating strings together.
5. name = "Mike"
6. print "Hello %s" % (name)
7. The % operator after a string is used to combine a string with variables. The% operator will replace a %s in the string with the string variable that comes after it.
8. **Instructions**
9. Take a look at the code in the editor. What do you think it'll do? Click Save & Submit when you think you know.

**String Formatting with %, Part 2**

Remember, we used the % operator to replace the %s placeholders with the variables in parentheses.

name = "Mike"

print "Hello %s" % (name)

You need the same number of %sterms in a string as the number of variables in parentheses:

print "The %s who %s %s!" % ("Knights", "say", "Ni")

*# This will print "The Knights who say Ni!"*

**Instructions**

Now it's your turn! We have \_\_\_ in the code to show you what you need to change!

1. Inside the string, replace the three\_\_\_ with %s.
2. After the string but before the three variables, replace the final \_\_\_with a %.
3. Hit *Save & Submit Code*.
4. Answer the questions in the console as they pop up! Type in your answer and hit Enter.

**And Now, For Something Completely Familiar**

Great job! You've learned a lot in this unit, including:

Three ways to create strings

'Alpha'

"Bravo"

str(3)

String methods

len("Charlie")

"Delta".upper()

"Echo".lower()

Printing a string

print "Foxtrot"

Advanced printing techniques

g = "Golf"

h = "Hotel"

print "%s, %s" % (g, h)

**Instructions**

Let's wrap it all up!

1. On [line 3](javascript:void(0)), create the variablemy\_string and set it to any string you'd like.
2. On [line 4](javascript:void(0)), print the length ofmy\_string.
3. On [line 5](javascript:void(0)), print the .upper() case version of my\_string.

**The datetime Library**

A lot of times you want to keep track of when something happened. We can do so in Python using datetime.

Here we'll use datetime to print the date and time in a nice format.

**Getting the Current Date and Time**

We can use a function calleddatetime.now() to retrieve the current date and time.

from datetime import datetime

print datetime.now()

The first line imports the datetimelibrary so that we can use it.

The second line will print out the current date and time.

**Instructions**

1. Create a variable called now and store the result of datetime.now() in it.
2. Then, print the value of now.

from datetime import datetime

now = datetime.now()

print now

**Extracting Information**

Notice how the output looks like 2013-11-25 23:45:14.317454. What if you don't want the entire date and time?

from datetime import datetime

now = datetime.now()

current\_year = now.year

current\_month = now.month

current\_day = now.day

You already have the first two lines.

In the third line, we take the year (and only the year) from the variable nowand store it in current\_year.

In the fourth and fifth lines, we store the month and day from now.

**Instructions**

1. On a new line, print now.year. Make sure you do it after setting thenow variable!
2. Then, print out now.month.
3. Finally, print out now.day.

**Hot Date**

What if we want to print today's date in the following format? mm/dd/yyyy. Let's use string substitution again!

from datetime import datetime

now = datetime.now()

print '%s-%s-%s' % (now.year, now.month, now.day)

*# will print: 2014-02-19*

Remember that the % operator will fill the %s placeholders in the string on the left with the strings in the parentheses on the right.

In the above example, we print 2014-02-19 (if today is February 19th, 2014), but you are going to print out02/19/2014.

**Instructions**

Print the current date in the form ofmm/dd/yyyy.

1. Change the string so that it uses a/ character in between the %splaceholders instead of a - character.
2. Re-arrange the parameters to the right of the % operator so that you print now.month, then now.day, thennow.year.

from datetime import datetime

now = datetime.now()

print '%s/%s/%s' % (now.month, now.day, now.year)

**Pretty Time**

Nice work! Let's do the same for the hour, minute, and second.

from datetime import datetime

now = datetime.now()

print now.hour

print now.minute

print now.second

In the above example, we just printed the current hour, then the current minute, then the current second.

We can again use the variable now to print the time.

**Instructions**

Similar to the last exercise, print the current time in the pretty form ofhh:mm:ss.

1. Change the string that you are printing so that you have a :character in between the %splaceholders.
2. Change the three things that you are printing from month, day, and year to now.hour, now.minute, andnow.second.

from datetime import datetime

now = datetime.now()

print '%s:%s:%s' % (now.hour, now.minute, now.second)

**Grand Finale**

We've managed to print the date and time separately in a very pretty fashion. Let's combine the two!

from datetime import datetime

now = datetime.now()

print '%s/%s/%s' % (now.month, now.day, now.year)

print '%s:%s:%s' % (now.hour, now.minute, now.second)

The example above will print out the date, then on a separate line it will print the time.

Let's print them all on the same line in a single print statement!

**Instructions**

Print the date and time together in the form: mm/dd/yyyy hh:mm:ss.

To start, change the format string to the left of the % operator.

1. Ensure that it has 6 %splaceholders.
2. Put slashes and colons and a space between the placeholders so that they fit the format above.

Then, change the variables in the parentheses to the right of the %operator.

1. Place the variables so that now.month, now.day, now.year are before now.hour, now.minute, now.second. Make sure that there is a ( before the six and a )after them.

from datetime import datetime

now = datetime.now()

print '%s/%s/%s %s:%s:%s' % (now.month, now.day, now.year,now.hour, now.minute, now.second)

**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 printsomething, 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 off what else is happening in the program.

**Instructions**

Check out the code in the editor. You'll see the type of program you'll be able to write once you've mastered control flow. Click Save & Submit to see what happens!

**Compare Closely!**

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

1. Equal to (==)
2. Not equal to (!=)
3. Less than (<)
4. Less than or equal to (<=)
5. Greater than (>)
6. Greater than or equal to (>=)

Comparators check if a value is (or is not) equal to, greater than (or equal to), or less than (or equal to) another value.

Note that == compares whether two things are equal, and = assigns a value to a variable.

**Instructions**

Set each variable to True or Falsedepending on what you think the result will be. For example, 1 < 2 will be True, because one is less than two.

1. Set bool\_one equal to the result of17 < 328
2. Set bool\_two equal to the result of100 == (2 \* 50)
3. Set bool\_three equal to the result of 19 <= 19
   1. Set bool\_four equal to the result of-22 >= -18
   2. Set bool\_five equal to the result of99 != (98 + 1)

Comparators check if a value is (or is not) equal to, greater than (or equal to), or less than (or equal to) another value.

Note that == compares whether two things are equal, and = assigns a value to a variable.

**Instructions**

Set each variable to True or Falsedepending on what you think the result will be. For example, 1 < 2 will be True, because one is less than two.

1. Set bool\_one equal to the result of17 < 328
2. Set bool\_two equal to the result of100 == (2 \* 50)
3. Set bool\_three equal to the result of 19 <= 19
   1. Set bool\_four equal to the result of-22 >= -18
   2. Set bool\_five equal to the result of99 != (98 + 1)

**Compare... Closelier!**

Excellent! It looks like you're comfortable with basic expressions and comparators.

But what about *extreme* expressions and comparators?

**Instructions**

Let's run through the comparators again with more complex expressions. Set each variable to True or Falsedepending on what you think the result will be.

1. Set bool\_one to the result of(20 - 10) > 15
2. Set bool\_two to the result of(10 + 17) == 3\*\*16
3. Set bool\_three to the result of1\*\*2 <= -1
4. Set bool\_four to the result of40 \* 4 >= -4
5. Set bool\_five to the result of 100!= 10\*\*2

**How the Tables Have Turned**

Comparisons result in either True or False, which are booleans as we learned before in [this exercise](http://www.codecademy.com/courses/introduction-to-python-6WeG3/0/3).

*# Make me true!*

bool\_one = 3 < 5

Let's switch it up: we'll give the boolean, and you'll write the expression, just like the example above.

**Instructions**

For each boolean value in the editor, write an expression that evaluates to that value.

Remember, comparators are: ==, !=,>, >=, <, and <=.

Use at least three different ones!

Don't just use True and False! That's cheating!

**To Be and/or Not to Be**

**Boolean operators** compare statements and result in boolean values. There are three boolean operators:

1. and, which checks if both the statements are True;
2. or, which checks if at least one of the statements is True;
3. not, which gives the opposite of the statement.

We'll go through the operators one by one.

**Instructions**

Look at the truth table in the editor. Don't worry if you don't completely get it yet—you will by the end of this section!

Click Save & Submit to continue.

**And**

The boolean operator and returnsTrue when the expressions on both sides of and are true. For instance:

* 1 < 2 and 2 < 3 is True;
* 1 < 2 and 2 > 3 is False.

**Instructions**

Let's practice with and. Assign each variable to the appropriate boolean value.

1. Set bool\_one equal to the result ofFalse and False
2. Set bool\_two equal to the result of-(-(-(-2))) == -2 and 4 >= 16\*\*0.5
3. Set bool\_three equal to the result of 19 % 4 != 300 / 10 / 10 and False
4. Set bool\_four equal to the result of-(1\*\*2) < 2\*\*0 and 10 % 10 <= 20 - 10\* 2
5. Set bool\_five equal to the result ofTrue and True

**Not**

The boolean operator not returnsTrue for false statements and Falsefor true statements.

For example:

* not False will evaluate to True, while not 41 > 40 will return False.

**Instructions**

Let's get some practice with not.

1. Set bool\_one equal to the result ofnot True
2. Set bool\_two equal to the result ofnot 3\*\*4 < 4\*\*3
3. Set bool\_three equal to the result of not 10 % 3 <= 10 % 2
4. Set bool\_four equal to the result ofnot 3\*\*2 + 4\*\*2 != 5\*\*2
5. Set bool\_five equal to the result ofnot not False

**This and That (or This, But Not That!)**

Boolean operators aren't just evaluated from left to right. Just like with arithmetic operators, there's an order of operations for boolean operators:

1. not is evaluated first;
2. and is evaluated next;
3. or is evaluated last.

For example, True or not False andFalse returns True. If this isn't clear, look at the Hint.

Parentheses () ensure your expressions are evaluated in the order you want. Anything in parentheses is evaluated as its own unit.

**Instructions**

Assign True or False as appropriate for bool\_one through bool\_five.

1. Set bool\_one equal to the result ofFalse or not True and True
2. Set bool\_two equal to the result ofFalse and not True or True
3. Set bool\_three equal to the result of True and not (False or False)
4. Set bool\_four equal to the result ofnot not True or False and not True
5. Set bool\_five equal to the result ofFalse or not (True and True)

**Mix 'n' Match**

Great work! We're almost done with boolean operators.

*# Make me false*

bool\_one = (2 <= 2) and "Alpha" == "Bravo"

**Instructions**

This time we'll give the expected result, and you'll use some combination of boolean operators to achieve that result.

Remember, the boolean operators areand, or, and not. Use each one at least once!

**What Good are Functions?**

You might have considered the situation where you would like to reuse a piece of code, just with a few different values. Instead of rewriting the whole code, it's much cleaner to define a **function**, which can then be used repeatedly.

**Instructions**

Check out the code in the editor. If you completed the [Tip Calculator][1] project, you'll remember going through and calculating tax and tip in one chunk of program. Here you can see we've defined two functions: taxto calculate the tax on a bill, and tipto compute the tip.

See how much of the code you understand at first glance (we'll explain it all soon). When you're ready, click Save & Submit to continue.

**Function Junction**

Functions are defined with three components:

1. The **header**, which includes the defkeyword, the name of the function, and any **parameters** the function requires. Here's an example:
2. def hello\_world(): // There are no parameters
3. An optional **comment** that explains what the function does.
4. """Prints 'Hello World!' to the console."""
5. The **body**, which describes the procedures the function carries out. The body is *indented*, just like for conditional statements.
6. print "Hello World!"

Here's the full function pieced together:

def hello\_world():

"""Prints 'Hello World!' to the console."""

print "Hello World!"

**Instructions**

Go ahead and create a function, spam, that prints the string "Eggs!" to the console. Don't forget to include a comment of your own choosing (enclose it in triple quotes!).

**Function Junction**

Functions are defined with three components:

1. The **header**, which includes the defkeyword, the name of the function, and any **parameters** the function requires. Here's an example:
2. def hello\_world(): // There are no parameters
3. An optional **comment** that explains what the function does.
4. """Prints 'Hello World!' to the console."""
5. The **body**, which describes the procedures the function carries out. The body is *indented*, just like for conditional statements.
6. print "Hello World!"

Here's the full function pieced together:

def hello\_world():

"""Prints 'Hello World!' to the console."""

print "Hello World!"

**Instructions**

Go ahead and create a function, spam, that prints the string "Eggs!" to the console. Don't forget to include a comment of your own choosing (enclose it in triple quotes!).

**Call and Response**

After defining a function, it must be**called** to be implemented. In the previous exercise, spam() in the last line told the program to look for the function called spam and execute the code inside it.

**Instructions**

We've set up a function, square. Call it on the number 10 (by putting 10between the parentheses of square()) on [line 9](javascript:void(0))!

**Parameters and Arguments**

Let's reexamine the first line that defined square in the previous exercise:

def square(n):

n is a **parameter** of square. A parameter acts as a variable name for a passed in **argument**. With the previous example, we called squarewith the argument 10. In this instance the function was called, n holds the value 10.

A function can require as many parameters as you'd like, but when you call the function, you should generally pass in a matching number of arguments.

**Instructions**

Check out the function in the editor,power. It should take two arguments, a base and an exponent, and raise the first to the power of the second. It's currently broken, however, because its parameters are missing.

Replace the \_\_\_s with the parametersbase and exponent and call power on a base of 37 and a power of 4.

def power(base,exponent): # Add your parameters here!

result = base\*\*exponent

print "%d to the power of %d is %d." % (base, exponent, result)

power(37,4) # Add your arguments here!

**Functions Calling Functions**

We've seen functions that can print text or do simple arithmetic, but functions can be much more powerful than that. For example, a function can call another function:

def fun\_one(n):

return n \* 5

def fun\_two(m):

return fun\_one(m) + 7

**Instructions**

Let's look at the two functions in the editor: one\_good\_turn (which adds 1to the number it takes in as an argument) and deserves\_another(which adds 2).

Change the body of deserves\_anotherso that it always adds 2 to the output of one\_good\_turn.

def one\_good\_turn(n):

return n + 1

def deserves\_another(n):

return one\_good\_turn(n) + 2

def cube(number):

return number\*number\*number

def by\_three(number):

if number % 3 == 0:

return cube(number)

else: return False

**I Know Kung Fu**

Remember import this from the first exercise in this course? That was an example of **import**ing a **module**. A module is a file that contains definitions—including variables and functions—that you can use once it is imported.

**Instructions**

Before we try any fancy importing, let's see what Python already knows about square roots. On [line 3](javascript:void(0)) in the editor, ask Python to

print sqrt(25)

which we would expect to equal five.

**Generic Imports**

Did you see that? Python said: "NameError: name 'sqrt' is not defined." Python doesn't know what square roots are—yet.

There is a Python module named maththat includes a number of useful variables and functions, and sqrt() is one of those functions. In order to access math, all you need is theimport keyword. When you simply import a module this way, it's called a**generic import**.

**Instructions**

You'll need to do two things here:

1. Type import math on [line 2](javascript:void(0)) in the editor.
2. Insert math. before sqrt() so that it has the form math.sqrt(). This tells Python not only to import math, but to get the sqrt() function from withinmath.

Then hit Save & Submit to see what Python now knows.

**Function Imports**

Nice work! Now Python knows how to take the square root of a number.

However, we only really needed thesqrt function, and it can be frustrating to have to keep typingmath.sqrt().

It's possible to import only certain variables or functions from a given module. Pulling in just a single function from a module is called a**function import**, and it's done with the from keyword:

from module import function

Now you can just type sqrt() to get the square root of a number—no more math.sqrt()!

**Instructions**

Let's import *only* the sqrt function from math this time. (You don't need the () after sqrt in the from mathimport sqrt bit.)

# Import \*just\* the sqrt function from math on line 3!

from math import sqrt

print sqrt(25)

**Universal Imports**

Great! We've found a way to handpick the variables and functions we want from modules.

What if we still want all of the variables and functions in a module but don't want to have to constantly typemath.?

**Universal import** can handle this for you. The syntax for this is:

from module import \*

**Instructions**

Use the power of from module import\* to import everything from the mathmodule on [line 3](javascript:void(0)) of the editor.

**[?](https://www.codecademy.com/courses/python-beginner-c7VZg/2/4?curriculum_id=4f89dab3d788890003000096)**

**[Stuck?](https://www.codecademy.com/courses/python-beginner-c7VZg/2/4?curriculum_id=4f89dab3d788890003000096)**[Get a hint!](https://www.codecademy.com/courses/python-beginner-c7VZg/2/4?curriculum_id=4f89dab3d788890003000096)

**Here Be Dragons**

Universal imports may look great on the surface, but they're not a good idea for one very important reason: they fill your program with a ton of variable and function names without the safety of those names still being associated with the module(s) they came from.

If you have a function of your very own named sqrt and you import math, your function is safe: there is yoursqrt and there is math.sqrt. If you dofrom math import \*, however, you have a problem: namely, two different functions with the exact same name.

Even if your own definitions don't directly conflict with names from imported modules, if you import \*from several modules at once, you won't be able to figure out which variable or function came from where.

For these reasons, it's best to stick with either import module and typemodule.name or just import specific variables and functions from various modules as needed.

**Instructions**

The code in the editor will show you everything available in the mathmodule.

Click Save & Submit Code to check it out (you'll see sqrt, along with some other useful things like pi, factorial, and [trigonometric functions](http://en.wikipedia.org/wiki/Trigonometry)).

**On Beyond Strings**

Now that you understand what functions are and how to import modules, let's look at some of the functions that are built in to Python (no modules required!).

You already know about some of the built-in functions we've used with strings, such as .upper(), .lower(),str(), and len(). These are great for doing work with strings, but what about something a little more analytic?

**Instructions**

What do you think the code in the editor will do? Click Save & Submit Code when you think you have an idea.

import math # Imports the math module

everything = dir(math) # Sets everything to a list of things from math

print everything # Prints 'em all!

**On Beyond Strings**

Now that you understand what functions are and how to import modules, let's look at some of the functions that are built in to Python (no modules required!).

You already know about some of the built-in functions we've used with strings, such as .upper(), .lower(),str(), and len(). These are great for doing work with strings, but what about something a little more analytic?

**Instructions**

What do you think the code in the editor will do? Click Save & Submit Code when you think you have an idea.

def biggest\_number(\*args):

print max(args)

return max(args)

def smallest\_number(\*args):

print min(args)

return min(args)

def distance\_from\_zero(arg):

print abs(arg)

return abs(arg)

biggest\_number(-10, -5, 5, 10)

smallest\_number(-10, -5, 5, 10)

distance\_from\_zero(-10)

10

-10

10

None

**max()**

The max() function takes any number of arguments and returns the largest one. ("Largest" can have odd definitions here, so it's best to usemax() on integers and floats, where the results are straightforward, and not on other objects, like strings.)

For example, max(1,2,3) will return 3(the largest number in the set of arguments).

**Instructions**

Try out the max() function on [line 3](javascript:void(0)) of the editor. You can provide any number of integer or float arguments to max().

# Set maximum to the max value of any set of numbers on line 3!

maximum = max(5,6,7,3,2,0)

print maximum

**min()**

min() then returns the smallest of a given series of arguments.

**Instructions**

Go ahead and set minimum equal to the min() of any set of integers or floats you'd like.

**abs()**

The abs() function returns the**absolute value** of the number it takes as an argument—that is, that number's distance from 0 on an imagined number line. For instance, 3and -3 both have the same absolute value: 3. The abs() function always returns a positive value, and unlikemax() and min(), it only takes a single number.

**Instructions**

Set absolute equal to the absolute value of -42 on [line 2](javascript:void(0)).

**type()**

Finally, the type() function returns the **type** of the data it receives as an argument. If you ask Python to do the following:

print type(42)

print type(4.2)

print type('spam')

Python will output:

<type 'int'>

<type 'float'>

<type 'str'>

**Instructions**

Have Python print out the type of anint, a float, and a str string in the editor. You can pick any values on which to call type(), so long as they produce one of each.

print type(87)

print type(7.653)

print type('somebody') 一定要加点

**Review: Functions**

Okay! Let's review functions.

def speak(message):

return message

if happy():

speak("I'm happy!")

elif sad():

speak("I'm sad.")

else:

speak("I don't know what I'm feeling.")

Again, the example code above is just there for your reference!

**Instructions**

1. First, def a function, shut\_down, that takes one argument s. Don't forget the parentheses or the colon!
2. Then, if the shut\_down function receives an s equal to "yes", it should return "Shutting down"
3. Alternatively, elif s is equal to"no", then the function should return"Shutdown aborted".
4. Finally, if shut\_down gets anything other than those inputs, the function should return "Sorry"

**[?](https://www.codecademy.com/courses/python-beginner-c7VZg/4/1?curriculum_id=4f89dab3d788890003000096)**

**[Hint](https://www.codecademy.com/courses/python-beginner-c7VZg/4/1?curriculum_id=4f89dab3d788890003000096)**

Ensure your function outputs appear exactly as shown!

Also, ensure your function returns the above values rather thanprinting them.

def shut\_down(s):

if s =="yes":

return "Shutting down"

elif s == "no":

return "Shutdown aborted"

else:

return "Sorry"

**Review: Modules**

Good work! Now let's see what you remember about importing modules (and, specifically, what's available in the math module).

**Instructions**

Import the math module in whatever way you prefer. Call its sqrt function on the number 13689 and print that value to the console.

**Before We Begin**

Let's first quickly review functions in Python.

def bigger(first, second):

print max(first, second)

return True

In the example above:

1. We define a function called biggerthat has two arguments called firstand second.
2. Then, we print out the larger of the two arguments using the built-in function max.
3. Finally, the bigger function returnsTrue.

Now try creating a function yourself!

**Instructions**

Write a function called answer that takes no arguments and returns the value 42.

Even without arguments, you will still need parentheses.

Don't forget the colon at the end of the function definition!

def answer():

return 42

没有argument 也要有()

**Planning Your Trip**

When planning a vacation, it's very important to know exactly how much you're going to spend.

def wages(hours):

*# If I make $8.35/hour...*

return 8.35 \* hours

The above example is just a refresher in how functions are defined.

Let's use functions to calculate your trip's costs.

**Instructions**

1. Define a function called hotel\_costwith one argument nights as input.
2. The hotel costs $140 per night. So, the function hotel\_cost should return140 \* nights.

def hotel\_cost(nights):

return 140\*nights

You're going to need to take a plane ride to get to your location.

def fruit\_color(fruit):

if fruit == "apple":

return "red"

elif fruit == "banana":

return "yellow"

elif fruit == "pear":

return "green"

1. The example above defines the function fruit\_color that accepts a string as the argument fruit.
2. The function returns a string if it knows the color of that fruit.

**Instructions**

1. Below your existing code, define a function called plane\_ride\_cost that takes a string, city, as input.
2. The function should return a different price depending on the location, similar to the code example above. Below are the valid destinations and their corresponding round-trip prices.

"Charlotte": 183  
"Tampa": 220  
"Pittsburgh": 222  
"Los Angeles": 475

def hotel\_cost(nights):

return 140\*nights

def plane\_ride\_cost(city):

if city == "Charlotte":

return 183

elif city == "Tampa":

return 220

elif city == "Pittsburgh":

return 222

elif city == "Los Angeles":

return 475

注意冒号啊

**Transportation**

You're also going to need a rental car in order for you to get around.

def finish\_game(score):

tickets = 10 \* score

if score >= 10:

tickets += 50

elif score >= 7:

tickets += 20

return tickets

In the above example, we first give the player 10 tickets for every point that the player scored. Then, we check the value of score multiple times.

1. First, we check if score is greater than or equal to 10. If it is, we give the player 50 bonus tickets.
2. If score is just greater than or equal to 7, we give the player 20 bonus tickets.
3. At the end, we return the total number of tickets earned by the player.

Remember that an elif statement is only checked if all preceding if/elifstatements fail.

**Instructions**

1. Below your existing code, define a function called rental\_car\_cost with an argument called days.
2. Calculate the cost of renting the car:
   * Every day you rent the car costs $40.
   * if you rent the car for 7 or more days, you get $50 off your total.
   * Alternatively (elif), if you rent the car for 3 or more days, you get $20 off your total.
   * You cannot get both of the above discounts.
3. Return that cost.

Just like in the example above, this check becomes simpler if you make the 7-day check an if statement and the 3-day check an elif statement.

**[?](https://www.codecademy.com/courses/python-beginner-P5YtY/1/3?curriculum_id=4f89dab3d788890003000096)**

**[Hint](https://www.codecademy.com/courses/python-beginner-P5YtY/1/3?curriculum_id=4f89dab3d788890003000096)**

Remember to use an elif so you don't subtract both of the discounts! (You'll want to do your check for >= 7 days first.)

def hotel\_cost(nights):

return 140\*nights

def plane\_ride\_cost(city):

if city == "Charlotte":

return 183

elif city == "Tampa":

return 220

elif city == "Pittsburgh":

return 222

elif city == "Los Angeles":

return 475

def rental\_car\_cost(days):

cost = days\*40

if days >= 7:

cost -= 50

elif days >= 3:

cost -= 20

return cost注意return 的位置啊

**Pull it Together**

Great! Now that you've got your 3 main costs figured out, let's put them together in order to find the total cost of your trip.

def double(n):

return 2 \* n

def triple(p):

return 3 \* p

def add(a, b):

return double(a) + triple(b)

1. We define two simple functions,double(n) and triple(p) that return 2 times or 3 times their input. Notice that they have n and p as their**arguments**.
2. We define a third function, add(a, b) that returns the sum of the previous two functions when called with a and b, respectively.

**Instructions**

1. Below your existing code, define a function called trip\_cost that takes two arguments, city and days.
2. Like the example above, have your function return the **sum** of calling therental\_car\_cost(days),hotel\_cost(days), andplane\_ride\_cost(city) functions.

It is completely valid to call thehotel\_cost(nights) function with the variable days. Just like the example above where we call double(n) with the variable a, we pass the value ofdays to the new function in the argument nights.

**Hey, You Never Know!**

You can't expect to only spend money on the plane ride, hotel, and rental car when going on a vacation. There also needs to be room for additional costs like fancy food or souvenirs.

**Instructions**

1. Modify your trip\_cost function definition. Add a third argument,spending\_money.
2. Modify what the trip\_costfunction does. Add the variablespending\_money to the sum that it returns.

def hotel\_cost(nights):

return nights\*140

print hotel\_cost

def plane\_ride\_cost(city):

if city == "Charlotte":

return 183

elif city == "Tampa":

return 220

elif city == "Pittsburgh":

return 222

elif city == "Los Angeles":

return 475

print plane\_ride\_cost

def rental\_car\_cost(days):

cost = days\*40

if days >= 7:

cost -= 50

elif days >= 3:

cost -= 20

return cost

print rental\_car\_cost

def trip\_cost(city,days,spending\_money):

return plane\_ride\_cost(city) + hotel\_cost(days) + rental\_car\_cost(days) + spending\_money

def hotel\_cost(nights):

return nights\*140

**Plan Your Trip!**

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**

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

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

**[?](https://www.codecademy.com/courses/python-beginner-P5YtY/1/6?curriculum_id=4f89dab3d788890003000096)**

**[Hint](https://www.codecademy.com/courses/python-beginner-P5YtY/1/6?curriculum_id=4f89dab3d788890003000096)**

You'll want to print the result of calling trip\_cost with the above values as arguments!

Your final call should look something like:

print trip\_cost("SOME CITY", NUM\_DAYS, SPENDING\_MONEY)

where SOME CITY is one of the four cities, NUM\_DAYS is the number of days and SPENDING\_MONEY is the amount of spending money.

def plane\_ride\_cost(city):

if city == "Charlotte":

return 183

elif city == "Tampa":

return 220

elif city == "Pittsburgh":

return 222

elif city == "Los Angeles":

return 475

def rental\_car\_cost(days):

cost = days\*40

if days >= 7:

cost -= 50

elif days >= 3:

cost -= 20

return cost

def trip\_cost(city,days,spending\_money):

return plane\_ride\_cost(city) + hotel\_cost(days) + rental\_car\_cost(days) + spending\_money

print trip\_cost("Los Angeles", 5, 600)

此处的print级别跨三个def,所以要打头写

**Introduction to Lists**

Lists are a **datatype** you can use to store a collection of different pieces of information as a sequence under a single variable name. (Datatypes you've already learned about include strings, numbers, and booleans.)

You can assign items to a list with an expression of the form

list\_name = [item\_1, item\_2]

with the items in between brackets. A list can also be empty: empty\_list =[].

Lists are very similar to strings, but there are a few key differences.

**Instructions**

The list zoo\_animals has three items (check them out on [line 1](javascript:void(0))). Go ahead and add a fourth! Just enter the name of your favorite animal (as a "string") on [line 1](javascript:void(0)), after the final comma but before the closing **]**.

zoo\_animals = ["pangolin", "cassowary", "sloth", "cat" ];

# One animal is missing!

if len(zoo\_animals) > 3:

print "The first animal at the zoo is the " + zoo\_animals[0]

print "The second animal at the zoo is the " + zoo\_animals[1]

print "The third animal at the zoo is the " + zoo\_animals[2]

print "The fourth animal at the zoo is the " + zoo\_animals[3]

**Access by Index**

You can access an individual item on the list by its **index**. An index is like an address that identifies the item's place in the list. The index appears directly after the list name, in between brackets, like this: list\_name[index].

**List indices begin with 0, not 1!** You access the first item in a list like this:list\_name[0]. The second item in a list is at index 1: list\_name[1]. Computer scientists love to start counting from zero.

**Instructions**

Write a statement that prints the result of adding the second and fourth items of the list. Make sure to access the list by index!

numbers = [5, 6, 7, 8]

print "Adding the numbers at indices 0 and 2..."

print numbers[0] + numbers[2]

print "Adding the numbers at indices 1 and 3..."

print numbers[1] + numbers[3]# Your code here!

**New Neighbors**

A list index behaves like any other variable name! It can be used to access as well as assign values.

You saw how to access a list index like this:

zoo\_animals[0]

*# Gets the value "pangolin"*

You can see how assignment works on[line 5](javascript:void(0)):

zoo\_animals[2] = "hyena"

*# Changes "sloth" to "hyena"*

**Instructions**

Write an assignment statement that will replace the item that currently holds the value "tiger" with another animal (as a string). It can be any animal you like.

zoo\_animals = ["pangolin", "cassowary", "sloth", "tiger"]

# Last night our zoo's sloth brutally attacked

#the poor tiger and ate it whole.

# The ferocious sloth has been replaced by a friendly hyena.

zoo\_animals[2] = "hyena"

# What shall fill the void left by our dear departed tiger?

# Your code here!

zoo\_animals[3] = "cat"

**Late Arrivals & List Length**

A list doesn't have to have a fixed length. You can add items to the end of a list any time you like!

letters = ['a', 'b', 'c']

letters.append('d')

print len(letters)

print letters

1. In the above example, we first create a list called letters.
2. Then, we add the string 'd' to the end of the letters list.
3. Next, we print out 4, the length of the letters list.
4. Finally, we print out ['a', 'b','c', 'd'].

**Instructions**

1. On [lines 5](javascript:void(0)), 6, and 7, append threemore items to the suitcase list, just like the second line of the example above. (Maybe bring a bathing suit?)
2. Then, set list\_length equal to the length of the suitcase list.

suitcase = []

suitcase.append("sunglasses")

# Your code here!

suitcase.append("books")

suitcase.append("dildoles")

suitcase.append("drugs")

list\_length =len(suitcase) # Set this to the length of suitcase

print "There are %d items in the suitcase." % (list\_length)

print suitcase

**List Slicing**

Sometimes, you only want to access a portion of a list.

letters = ['a', 'b', 'c', 'd', 'e']

slice = letters[1:3]

print slice

print letters

1. In the above example, we first create a list called letters.
2. Then, we take a subsection and store it in the slice list. We start at the index before the colon and continue up to but not including the index after the colon.
3. Next, we print out ['b', 'c']. Remember that we start counting indices from 0 and that we stoppedbefore index 3.
4. Finally, we print out ['a', 'b','c', 'd', 'e'], just to show that we did not modify the original letterslist.

**Instructions**

1. On [line 4](javascript:void(0)), create a list calledmiddle containing only the two middle items from suitcase.
2. On [line 5](javascript:void(0)), create a list called lastmade up only of the last two items from suitcase.

suitcase = ["sunglasses", "hat", "passport", "laptop", "suit", "shoes"]

first = suitcase[0:2] # The first and second items (index zero and one)

middle = suitcase[2:4] # Third and fourth items (index two and three)

last = suitcase[4:6] # The last two items (index four and five)

index4 5 就是 4:6

**Slicing Lists and Strings**

You can slice a string exactly like a list! In fact, you can think of strings as lists of characters: each character is a sequential item in the list, starting from index 0.

my\_list[:2]

*# Grabs the first two items*

my\_list[3:]

*# Grabs the fourth through last items*

If your list slice includes the very first or last item in a list (or a string), the index for that item doesn't have to be included.

**Instructions**

1. Assign to dog a slice of animalsfrom index 3 up until *but not including*index 6.
2. Assign to frog a slice of animalsfrom index 6 until the end of the string.

animals = "catdogfrog"

cat = animals[:3] # The first three characters of animals

dog = animals[3:6] # The fourth through sixth characters

frog = animals[6:] # From the

**Maintaining Order**

Sometimes you need to search for an item in a list.

animals = ["ant", "bat", "cat"]

print animals.index("bat")

1. First, we create a list called animalswith three strings.
2. Then, we print the first index that contains the string "bat", which will print 1.

We can also insert items into a list.

animals.insert(1, "dog")

print animals

1. We insert "dog" at index 1, which moves everything down by 1.
2. We print out ["ant", "dog", "bat", "cat"]

**Instructions**

1. Use the .index(item) function to find the index of "duck". Assign that result to a variable called duck\_index.
2. Then .insert(index, item) the string "cobra" at that index.

animals = ["aardvark", "badger", "duck", "emu", "fennec fox"]

duck\_index = animals.index("duck") # Use index() to find "duck"

print duck\_index

# Your code here!

animals.insert(duck\_index,"cobra")

print animals # Observe what prints after the insert operation

**For One and All**

If you want to do something with every item in the list, you can use a forloop. If you've learned about forloops in JavaScript, pay close attention! They're different in Python.

for variable in list\_name:

*# Do stuff!*

A variable name follows the forkeyword; it will be assigned the value of each list item in turn.

Then in list\_name designateslist\_name as the list the loop will work on. The line ends with a colon (:) and the indented code that follows it will be executed once per item in the list.

**Instructions**

Write a statement in the indented part of the for-loop that prints a number equal to 2 \* number for every list item.

my\_list = [1,9,3,8,5,7]

for number in my\_list:

print 2\*number# Your code here

**More with 'for'**

If your list is a jumbled mess, you may need to sort() it.

animals = ["cat", "ant", "bat"]

animals.sort()

for animal in animals:

print animal

1. First, we create a list called animalswith three strings. The strings are not in alphabetical order.
2. Then, we sort animals into alphabetical order. Note that .sort()modifies the list rather than returning a new list.
3. Then, for each item in animals, we print that item out as "ant", "bat", "cat" on their own line each.

**Instructions**

1. Write a for-loop that iterates overstart\_list and .append()s each number squared (x \*\* 2) tosquare\_list.
2. Then sort square\_list!

start\_list = [5, 3, 1, 2, 4]

square\_list = []

for nm in start\_list:

square\_list.append(nm\*\*2)此刻用的是square\_list, 直接引用

square\_list.sort()

print square\_list

**This Next Part is Key**

A dictionary is similar to a list, but you access values by looking up a **key** instead of an index. A key can be any string or number. Dictionaries are enclosed in curly braces, like so:

d = {'key1' : 1, 'key2' : 2, 'key3' : 3}

This is a dictionary called d with three **key-value pairs**. The key 'key1'points to the value 1, 'key2' to 2, and so on.

Dictionaries are great for things like phone books (pairing a name with a phone number), login pages (pairing an e-mail address with a username), and more!

**Instructions**

Print the values stored under the'Sloth' and 'Burmese Python' keys. Accessing dictionary values by key is just like accessing list values by index:

residents['Puffin']

*# Gets the value 104*

Check the Hint if you need help!

**# Assigning a dictionary with three key-value pairs to residents:**

**residents = {'Puffin' : 104, 'Sloth' : 105, 'Burmese Python' : 106}**

**print residents['Puffin'] # Prints Puffin's room number**

**print residents['Sloth']# Your code here!**

**print residents['Burmese Python']**

**New Entries**

Like Lists, Dictionaries are "mutable". This means they can be changed after they are created. One advantage of this is that we can add new key/value pairs to the dictionary after it is created like so:

dict\_name[new\_key] = new\_value

An empty pair of curly braces {} is an empty dictionary, just like an empty pair of [] is an empty list.

The length len() of a dictionary is the number of key-value pairs it has. Each pair counts only once, even if the value is a list. (That's right: you can put listsinside dictionaries!)

**Instructions**

Add at least three more key-value pairs to the menu variable, with the dish name (as a "string") for the key and the price (a float or integer) as the value. Here's an example:

menu['Spam'] = 2.50

**[?](https://www.codecademy.com/courses/python-beginner-en-pwmb1/2/2?curriculum_id=4f89dab3d788890003000096)**

**[Stuck?](https://www.codecademy.com/courses/python-beginner-en-pwmb1/2/2?curriculum_id=4f89dab3d788890003000096)**[Get a hint!](https://www.codecademy.com/courses/python-beginner-en-pwmb1/2/2?curriculum_id=4f89dab3d788890003000096)

menu = {} # Empty dictionary

menu['Chicken Alfredo'] = 14.50 # Adding new key-value pair

print menu['Chicken Alfredo']

# Your code here: Add some dish-price pairs to menu!

menu['dish-price pairs']= 3

menu['guns'] = 50

menu['whore'] = 6

print "There are " + str(len(menu)) + " items on the menu."

print menu

**Changing Your Mind**

Because dictionaries are mutable, they can be changed in many ways. Items can be removed from a dictionary with the del command:

del dict\_name[key\_name]

will remove the key key\_name and its associated value from the dictionary.

A new value can be associated with a key by assigning a value to the key, like so:

dict\_name[key] = new\_value

**Instructions**

Delete the 'Sloth' and 'Bengal Tiger' items from zoo\_animals usingdel.

Set the value associated with'Rockhopper Penguin' to anything other than 'Arctic Exhibit'.

# key - animal\_name : value - location

zoo\_animals = { 'Unicorn' : 'Cotton Candy House',

'Sloth' : 'Rainforest Exhibit',

'Bengal Tiger' : 'Jungle House',

'Atlantic Puffin' : 'Arctic Exhibit',

'Rockhopper Penguin' : 'Arctic Exhibit'}

# A dictionary (or list) declaration may break across multiple lines

# Removing the 'Unicorn' entry. (Unicorns are incredibly expensive.)

del zoo\_animals['Unicorn']

del zoo\_animals['Sloth']

del zoo\_animals['Bengal Tiger']# Your code here!

zoo\_animals['Rockhopper Penguin'] ='Rainforest Exhibit'

print zoo\_animals

**Remove a Few Things**

Sometimes you need to remove something from a list.

beatles = ["john","paul","george","ringo","stuart"]

beatles.remove("stuart")

print beatles

>> ["john","paul","george","ringo"]

1. We create a list called beatles with 5 strings.
2. Then, we remove the first item from beatles that matches the string"stuart". Note that .remove(item)does not return anything.
3. Finally, we print out that list just to see that "stuart" was actually removed.

**Instructions**

Remove 'dagger' from the list of items stored in the backpack variable.

backpack = ['xylophone', 'dagger', 'tent', 'bread loaf']

backpack.remove('dagger') 看到这里了没，必须是小括号

print backpack

**It's Dangerous to Go Alone! Take This**

Let's go over a few last notes about**dictionaries**

my\_dict = {

"fish": ["c", "a", "r", "p"],

"cash": -4483,

"luck": "good"

}

print my\_dict["fish"][0]

1. In the example above, we created a dictionary that holds many types of values.
2. The key "fish" has a list, the key"cash" has an int, and the key "luck"has a string.
3. Finally, we print the letter 'c'. When we access a value in the dictionary like my\_dict["fish"], we have direct access to that value. So we can access the item at index '0' in the list stored by the key "fish"

**Instructions**

1. Add a key to inventory called'pocket'
2. Set the value of 'pocket' to be a list consisting of the strings'seashell', 'strange berry', and'lint'
3. .sort() the items in the list stored under the 'backpack' key
4. Then .remove('dagger') from the list of items stored under the'backpack' key
5. Add 50 to the number stored under the 'gold' key
6. You can use list functions with a list stored in a dictionary as follows:
7. dict\_name['list\_key'].list\_function()
8. This should help you delete'dagger' from the list of items stored under the 'backpack' key. (You can use .remove() on lists just like you can on dictionaries.)

inventory = {

'gold' : 500,

'pouch' : ['flint', 'twine', 'gemstone'], # Assigned a new list to 'pouch' key

'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']

}

# Adding a key 'burlap bag' and assigning a list to it

inventory['burlap bag'] = ['apple', 'small ruby', 'three-toed sloth']

# Sorting the list found under the key 'pouch'

inventory['pouch'].sort()

# Your code here

inventory['pocket'] = ['seashell', 'strange berry','lint']

inventory['backpack'].sort()

inventory['backpack'].remove('dagger')

inventory['gold'] = 500+50

**BeFOR We Begin**

Before we begin our exercise, we should go over the Python for loop one more time. For now, we are only going to go over the for loop in terms of how it relates to lists and dictionaries. We'll explain more coolfor loop uses in later courses.

for loops allow us to iterate through all of the elements in a list from the left-most (or zeroth element) to the right-most element. A sample loop would be structured as follows:

a = ["List of some sort”]

for x in a:

# Do something for every x

This loop will run all of the code in the indented block under the for x in a:statement. The item in the list that is currently being evaluated will be x. So running the following:

for item in [1, 3, 21]:

print item

would print 1, then 3, and then 21. The variable between for and in can be set to any variable name (currentlyitem), but you should be careful to avoid using the word “list” as a variable, since that's a reserved word (that is, it means something special) in the Python language.

**说明**

Use a for loop to print out all of the elements in the list names.

names = ["Adam","Alex","Mariah","Martine","Columbus"]

for ui in names:

print ui

**This is KEY! 不仅是list,也可以是key**

You can also use a for loop on a dictionary to loop through its *keys* with the following:

*# A simple dictionary*

d = {"foo" : "bar"}

for key in d:

print d[key] *# prints "bar"*

Note that dictionaries are **unordered**, meaning that any time you loop through a dictionary, you will go through *every* key, but you are not guaranteed to get them in any particular order.

**说明**

Use a for loop to go through thewebster dictionary and print out all of the definitions.

webster = {

"Aardvark" : "A star of a popular children's cartoon show.",

"Baa" : "The sound a goat makes.",

"Carpet": "Goes on the floor.",

"Dab": "A small amount."

}

for key in webster:

print webster[key]

# Add your code below!

如果不想print key怎么办

**Control Flow and Looping**

The blocks of code in a for loop can be as big or as small as they need to be.

While looping, you may want to perform different actions depending on the particular item in the list.

numbers = [1, 3, 4, 7]

for number in numbers:

if number > 6:

print number

print "We printed 7."

1. In the above example, we create a list with 4 numbers in it.
2. Then we loop through the numberslist and store each item in the list in the variable number.
3. On each loop, if number is greater than 6, we print it out. So, we print7.
4. Finally, we print out a sentence.

Make sure to keep track of your indentation or you may get confused!

**说明**

1. Like step 2 above, loop through each item in the list called a.
2. Like step 3 above, if the number is even, print it out. You can test ifthe item % 2 == 0 to help you out.

a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

for n in a:

if n % 2 == 0:

print n

**Lists + Functions**

Functions can also take lists as inputs and perform various operations on those lists.

def count\_small(numbers):

total = 0

for n in numbers:

if n < 10:

total = total + 1

return total

lost = [4, 8, 15, 16, 23, 42]

small = count\_small(lost)

print small

1. In the above example, we define a function count\_small that has one argument, numbers.
2. We initialize a variable total that we can use in the for loop.
3. For each item n in numbers, if n is less than 10, we increment total.
4. After the for loop, we return total.
5. After the function definition, we create an array of numbers called lost.
6. We call the count\_small function, pass in lost, and store the returned result in small.
7. Finally, we print out the returned result, which is 2 since only 4 and 8are less than 10.

**说明**

Write a function that counts how many times the string "fizz" appears in a list.

1. Write a function called fizz\_countthat takes a list x as input.
2. Create a variable count to hold the ongoing count. Initialize it to zero.
3. for each item in x:, if that item is equal to the string "fizz" then increment the count variable.
4. After the loop, please return thecount variable.

For example,fizz\_count(["fizz","cat","fizz"])should return 2.

# Write your function below!

def fizz\_count(x):

count = 0

for m in x:

if m == "fizz":

count = count + 1

return count

注意空格

**String Looping**

As we've mentioned, strings are like lists with characters as elements. You can loop through strings the same way you loop through lists! While we won't ask you to do that in this section, we've put an example in the editor of how looping through a string might work.

for letter in "Codecademy":

print letter

# Empty lines to make the output pretty

print

print

word = "Programming is fun!"

for letter in word:

# Only print out the letter i

if letter == "i":

print letter

**Your Own Store!**

Okay—on to the core of our project.

Congratulations! You are now the proud owner of your very own Codecademy brand supermarket.

animal\_counts = {

"ant": 3,

"bear": 6,

"crow": 2

}

In the example above, we create a new dictionary called animal\_counts with three entries. One of the entries has the key "ant" and the value 3.

**说明**

1. Create a new dictionary called prices using {} format like the example above.
2. Put these values in your prices dictionary, in between the {}:

"banana": 4,

"apple": 2,

"orange": 1.5,

"pear": 3

Yeah, this place is really expensive. (Your supermarket subsidizes the zoo from the last course.)

prices = {"banana": 4,"apple": 2,"orange": 1.5, "pear": 3}

**Investing in Stock**

Good work! As a store manager, you’re also in charge of keeping track of your stock/inventory.

**说明**

Create a stock dictionary with the values below.

"banana": 6

"apple": 0

"orange": 32

"pear": 15

**Keeping Track of the Produce**

Now that you have all of your product info, you should print out all of your inventory information.

once = {'a': 1, 'b': 2}

twice = {'a': 2, 'b': 4}

for key in once:

print "Once: %s" % once[key]

print "Twice: %s" % twice[key]

1. In the above example, we create two dictionaries, once and twice, that have the same keys.
2. Because we know that they have the same keys, we can loop through one dictionary and print values from both once and twice.

**说明**

1. Loop through each key in prices.
2. Like the example above, for each key, print out the key along with its price and stock information. Print the answer in the following format:

apple

price: 2

stock: 0

Like the example above, because you know that the prices and stockdictionary have the same keys, you can access the stock dictionary while you are looping through prices.

When you're printing, you can use the syntax from the example above.

rices = {"banana": 4,"apple": 2,"orange": 1.5, "pear": 3}

stock = {"banana": 6,"apple": 0,"orange": 32,"pear": 15}

for key in prices:

print key

print "price: %s" % prices[key]

print "stock: %s" % stock[key]

**Something of Value**

For paperwork and accounting purposes, let's record the total value of your inventory. It's nice to know what we're worth!

**说明**

Let's determine how much money you would make if you sold all of your food.

1. Create a variable called total and set it to zero.
2. Loop through the pricesdictionaries.
3. For each key in prices, multiply the number in prices by the number in stock. Print that value into the console and then add it to total.
4. Finally, outside your loop, print total.

**[?](https://www.codecademy.com/zh/courses/python-beginner-en-IZ9Ra/1/4?curriculum_id=4f89dab3d788890003000096)**

**[提示](https://www.codecademy.com/zh/courses/python-beginner-en-IZ9Ra/1/4?curriculum_id=4f89dab3d788890003000096)**

The value of any given product is its number of items in stock multiplied by its price. For instance, the total cost for bananas would be 24 (a price of 4 multiplied by 6 bananas in stock).

You should print the number all by itself—no need for any additional text!

prices = {

"banana" : 4,

"apple" : 2,

"orange" : 1.5,

"pear" : 3,

}

stock = {

"banana" : 6,

"apple" : 0,

"orange" : 32,

"pear" : 15,

}

for key in prices:

print key

print "price: %s" % prices[key]

print "stock: %s" % stock[key]

total = 0

for key in prices:

total = total + (prices[key] \* stock[key])

print total

**Shopping at the Market**

Great work! Now we're going to take a step back from the management side and take a look through the eyes of the shopper.

In order for customers to order online, we are going to have to make a consumer interface. Don't worry: it's easier than it sounds!

**Instructions**

First, make a **list** called groceries with the values "banana","orange", and"apple".

groceries = ["banana", "orange", "apple"]

**Making a Purchase**

Good! Now you're going to need to know how much you’re paying for all of the items on your grocery list.

def sum(numbers):

total = 0

for number in numbers:

total += number

return total

n = [1, 2, 5, 10, 13]

print sum(n)

1. In the above example, we first define a function called sum with an argument numbers.
2. We initialize the variable total that we will use as our running sum.
3. For each number in the list, we add that number to the running sum total.
4. At the end of the function, we return the running sum.
5. After the function, we create, n, a list of numbers.
6. Finally, we call the sum(numbers)function with the variable n and print the result.

**Instructions**

1. Define a function compute\_billthat takes one argument food as input.
2. In the function, create a variabletotal with an initial value of zero.
3. For each item in the food list, add the price of that item to total.
4. Finally, return the total.

Ignore whether or not the item you're billing for is in stock.

Note that your function should work for **any** food list.

shopping\_list = ["banana", "orange", "apple"]

stock = {

"banana": 6,

"apple": 0,

"orange": 32,

"pear": 15

}

prices = {

"banana": 4,

"apple": 2,

"orange": 1.5,

"pear": 3

}

# Write your code below!

def compute\_bill(food):

total =0

for item in food:

total=total+prices[item]

return total

print compute\_bill(shopping\_list)

**Stocking Out**

Now you need your compute bill function to take the stock/inventory of a particular item into account when computing the cost.

Ultimately, if an item isn't in stock, then it shouldn't be included in the total. You can't buy or sell what you don't have!

**Instructions**

Make the following changes to your compute\_bill function:

1. While you loop through each item of food, only add the price of the item to total if the item's stock count is greater than zero.
2. If the item is in stock and after you add the price to the total, subtract one from the item's stock count.

shopping\_list = ["banana", "orange", "apple"]

stock = {

"banana": 6,

"apple": 0,

"orange": 32,

"pear": 15

}

prices = {

"banana": 4,

"apple": 2,

"orange": 1.5,

"pear": 3

}

# Write your code below!

def compute\_bill(food):

total = 0

for item in food:

if stock[item]>0:

total = total + prices[item]

stock[item] = stock[item] - 1

return total

print total

**Lesson Number One**

Welcome to this "Challenge Course". Until now we've been leading you by the hand and working on some short and relatively easy projects. This is a**challenge** so be ready. We have faith in you!

We’re going to switch it up a bit and allow you to be the teacher of your own class. Make a gradebook for all of your students.

animal\_sounds = {

"cat": ["meow", "purr"],

"dog": ["woof", "bark"],

"fox": [],

}

print animal\_sounds["cat"]

The example above is just to remind you how to create a dictionary and then to access the item stored by the"cat" key.

**Instructions**

1. Create three dictionaries: lloyd,alice, and tyler.
2. Give each dictionary the keys"name", "homework", "quizzes", and"tests".
3. Have the "name" key be the name of the student (that is, lloyd's name should be "Lloyd") and the other keys should be an empty list. (We'll fill in these lists soon!)

Hi Alice, a few hours later, it would be a new year. HAPPY NEW YEAR, HAVE MORE IDEAS, SUCCESS, LOVE AND NEVER REGRET. IT WAS A GREAT PLEASURE TO MEET U.