## -Web Developer Skills

>>Learn to build professional websites and applications as used by real businesses.

-Make a Website
-Make an Interactive Wesite
-Learn Sass
-Deploy a Website
-Learn Ruby on Rails
-Learn AngularJS 1.X
-Introduction to React.js:Part 1 (Preview)
-Introduction to REact.js:Part 2 (Preview)
-Ruby on Rails : Authentication
-Learn the Command Line
-Learn Git
-Learn SQL
-SQL : Table Transformation
-SQL : Analyzing Business Metrics
-Learn Java

## -Language Skills

>> Learn core programming concepts and syntax for the world's most popular languages.

-HTML & CSS
-JavaScript
-jQuery
-PHP
-Python
-Ruby

## -Goals

>> Get started with coding with these 30 minute goals.

-Animate your Name
-About you
-Sun, Earth, and Code

## -APIs

>> Learn how to use popular APIs to make your own applications.

-Learn the Watson API

## -Web Developer Skills

>>Learn to build professional websites and applications as used by real businesses.

### -Make a Website
Explore HTML & CSS fundamentals as you build a website in this introductory course to web development.

### -Make an Interactive Wesite
Build the Flipboard home page and learn how to add interactivity to your website.

### -Learn Sass
Expand your CSS knowledge by learning SCSS syntax, nesting, functions, and more in this course on the Sass styling language.

### -Deploy a Website
Learn how to publish a personal website to the public Internet.

### -Learn Ruby on Rails
Learn the basics of building applications with this convenient, powerful web development framework.

### -Learn AngularJS 1.X
Learn how to easily build single-page web applications using this popular JavaScript framework.

### -Introduction to React.js:Part 1 (Preview)
Build powerful interactive applications with this popular JavaScript library

### -Introduction to REact.js:Part 2 (Preview)
Build powerful interactive applications with this popular JavaScript library

### -Ruby on Rails : Authentication
Learn how to add user sign up, login and logout functionality to your Rails applications in this intermediate course.

### -Learn the Command Line
Discover the power of this simple, yet essential text-based tool and increase your productivity as a developer.

### -Learn Git
Learn to sve and manage different versions of your code projects with this essential tool.

### -Learn SQL
Learn to communicate with databases using SQL, the standard data-management language

### -SQL : Table Transformation
Practice more SQL in this course that covers how to manipulate and transform data.

### -SQL : Analyzing Business Metrics
Explore SQL futher in this course focusing on how to analyze data.

### -Learn Java
Learn the basics of the popular Java language in this introductory course.

## -Language Skills

>> Learn core programming concepts and syntax for the world's most popular languages.

### -HTML & CSS
Learn how to create websites by structuring and styling your pages with HTML and CSS.

### -JavaScript
Learn the fundamentals of JavaScript, the programming language of the Web.

### -jQuery
Learn how to make your websites interactive and create animations by using jQuery.

### -PHP
Learn to program in PHP, a widespread language that powers sites like Facebook.

### -Python
Learn to program in Python, a powerful language used by sites like YouTube and Dropbox.

### -Ruby
Learn to program in Ruby, a flexible language used to create sites like Codecademy.

## -Goals

>> Get started with coding with these 30 minute goals.

### -Animate your Name
Create an animation of your name

### -About you
Make a website all about you

### -Sun, Earth, and Code
Build your own galaxy

## -APIs

>> Learn how to use popular APIs to make your own applications.

### -Learn the Watson API
Use IBM’s Personality Insights API to analyze traits shared between two Twitter users.

Codecademy

## Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

Lesson : Python Syntax
This lesson will introduce you to Python, a general-purpose, object-oriented language you can use for countless stardalonone projects or scripting applications.

Lesson : Tip Calculator
Now that you've completed the lesson on Python syntax, let's see if you can put your newfound skills to use. In this lesson, you'll create a simple calculator that determines the price of a meal after tax and tip.

## Unit 2 : Strings and Console Output

Lesson : Strings & Console Output
This lesson will introduce you to strings and console output in Python, including creating string literals, calling a variety of string methods, and using the "print" keyword.

Lesson : Date and Time
This lesson is a follow up to Unit 2 : Strings and Console input and will give you practice with the concepts introduced in that lesson.

## Unit 3 : Conditionals and Control Flow

Lesson : Conditionals & Control Flow
In this lesson, we'll learn how to create programs that generate different outcomes based on user input!

Lsesson : PygLatin
In this lesson we'll put together all of the Python skills we've learned so far including string manipulation and branching. We'll be building a Pyg Latin translator. (That's Pig Latin for Python Programmers!)

## Unit 4 : Functions

Lesson : Functions
A function is a reusable section of code written to perform a specific task in a program. We gave you a taste of functions in Unit 3; here, you'll learn how to create your own.

Lesson : Taking a Vacation
Hard day at work? Tough day at school? Take a load off with a programming vacation!

## Unit 5 : Lists & Dictionaries

Lesson : Python Lists and Dictionaries
Lists and dictionaries are powerful tools you can use to store, organize, and manipulate all kinds of information.

Lesson : A Day at the Supermarket
Let's manage our own supermarket and buy some goods along the way!

## Unit 6 : Student Becomes the Teacher

Lesson : Student Becomes the Teacher
Use what you've learned so far to manage your own class.

## Unit 7 : Lists and Functions

Lesson : Lists and Functions
Now that you've learned about lists, let's turbo-charge them with functions.

Lesson : Battleship!
In this lesson, we will make a simplified version of the classic board game Battleship!
We'll use functions, lists, and conditionals to make our game.

## Unit 8 : Loops

Lesson : Loops
Loops allow you to quickly iterate over information in Python. In this lesson, we'll cover two types of loop : 'while' and 'for'

Lesson : Practice Makes Perfect
You know a lot of Python now. Let's do some practice problems!

## Unit 9 : Exam. Statistics

Lesson : Exam Statistics
Your students just took their first test. It's time to see how everyone did. Let's write a program to compute the mean, variance, and standard deviation of the test scores.

## Unit 10 : Advanced Topics in Python

Lesson : Advanced Topics in Python
In this lesson, we'll cover some of the more complex aspects of Python, including iterating over data structures, list comprehensions, list slicing, and lambda expressions.

Lesson : Introdcution to Bitwise Operators
Bitwise operations directly manipulate bits--patterns of 0s and 1s. Though they can be tricky to learn at fi rst, their speed makes them a useful addition to any programmer's toolbox.

## Unit 11 : Introduction to Classes

Lesson : Introduction to Classes
Classes are a crucial part of object-oriented programming (OOP). In this lesson, we'll explain what classes are, why they're important, and how to use them effectively.

Lesson : Classes
Make your own Car and learn how to driveCar()!

## Unit 12 : File input and Output

Lesson : File Input/ Output
Now that you understand Python syntax and have been introduced to some Python best practices, let's apply what you've lear ned to a real-world application: writing data to file.

Codecademy
# Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

### Python Syntax

Variables and Data Types
Whitespace and Statements
Comments
Math Operations

practice) Paint the bill

## Unit 2 : Strings and Console Output

### Strings and Console Output

Strings
String Methods
Print
Advanced Printing

practice) Date and Time

## Unit 3 : Conditionals and Control Flow

### Conditionals & Control Flow

Introduction to Control Flow
Comparators
Boolean Operators
Else and Elif

practice) PygLatin

## Unit 4 : Functions

### Functions

Introduction to Functions
Function Syntax
Importing Modules
Built-In Functions

practice) Taking a Vacation

## Unit 5 : Lists & Dictionaries

### Python Lists and Dictionaries

Lists
Lists Capabilities and Functions
Dictionaries

practice) A Day at the Supermarket

## Unit 6 : Student Becomes the Teacher

### Student Becomes the Teacher

Good Morning Class!
Just Average

## Unit 7 : Lists and Functions

### Lists and Functions

List Recap
Function Recap
Introdcution to Using Funcions With Lists
Using the Entire List in a Function
Using Lists of Lists in Functions

practice) Battleship!

## Unit 8 : Loops

### Loops

While Loops
For Loops
Step Up Your 'For's

practice) Practice Makes Perfect

## Unit 9 : Exam Statistics

### Exam Statistics

Review
The Average Grade
Do the grades vary?

## Unit 10 : Advanced Topics in Python

### Advanced Topics in Python

Iteration Nation
List Comprehensions
List Slicing
Lambdas

### Introduction to Bitwise Operators

Binary Representation
The Bitwise Operators
A Bit More Complicated

## Unit 11 : Introduction to Classes

### Introduction to Classes

Class Basics
Member Variables and Functions
Inheritance

## Unit 12 : File Input and Output

### File Input and Output

Introduction to File I/O
The Devil's in the Details

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

### Python Syntax

- Variables and Data Types
- Whitespace and Statements
- Comments
- Math Operations

practice) Paint the bill

## Unit 2 : Strings and Console Output

### Strings and Console Output

- Strings
- String Methods
- Print
- Advanced Printing

practice) Date and Time

## Unit 3 : Conditionals and Control Flow

### Conditionals & Control Flow

- Introduction to Control Flow
- Comparators
- Boolean Operators
- Else and Elif

practice) PygLatin

## Unit 4 : Functions

### Functions

- Introduction to Functions
- Function Syntax
- Importing Modules
- Built-In Functions

practice) Taking a Vacation

## Unit 5 : Lists & Dictionaries

### Python Lists and Dictionaries

- Lists
- Lists Capabilities and Functions
- Dictionaries

practice) A Day at the Supermarket

## Unit 6 : Student Becomes the Teacher

### Student Becomes the Teacher

- Good Morning Class!
- Just Average

## Unit 7 : Lists and Functions

### Lists and Functions

- List Recap
- Function Recap
- Introdcution to Using Functions with Lists
- Using the Entire List in a Function
- Using Lists of Lists in Functions

practice) Battleship!

## Unit 8 : Loops

### Loops

- While Loops
- For Loops
- Step Up Your 'For's

practice) Practice Makes Perfect

## Unit 9 : Exam Statistics

### Exam Statistics

- Review
- The Average Grade
- Do the grades vary?

## Unit 10 : Advanced Topics in Python

### Advanced Topics in Python

- Iteration Nation
- List Comprehensions
- List Slicing
- Lambdas

### Introduction to Bitwise Operators

- Binary Representation
- The Bitwise Operators
- A Bit More Complicated

## Unit 11 : Introduction to Classes

### Introduction to Classes

- Class Basics
- Member Variables and Functions
- Inheritance

## Unit 12 : File Input and Output

### File Input and Output

- Introduction to File I/O
- The Devil's in the Details

Codecademy
# Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

### Python Syntax

**Variables and Data Types**
1. Welcome!
2. Variables
3. Booleans
4. You've Been Reassigned

**Whitespace and Statements**
5. Whitespace
6. Whitesapce Means Right Space
7. A Matter of Interpretation

**Comments**
8. Single Line Comments
9. Multi-Line Comments

**Math Operations**
10. Math
11. Exponentiation
12. Modulo

**Review**
13. Bringing it All Together

### Tip Calculator

**Paint the bill**
1. The Meal
2. The Tax
3. The Tip
4. Reassign in a Single Line
5. The Total

## Unit 2 : Strings and Console Output

### Strings and Console Output

**Stings**
1. Strings
2. Practice
3. Escaping Characters
4. Access by Index

**String Methods**
5. String methods
6. lower ( )
7. upper ( )
8. str ( )
9. Dot Notation

**Print**
10. Printing Strings
11. Printing Variables

**Advanced Printing**
12. String Concatenation
13. Explicit String Conversion
14. String Formatting with %, Part 1
15. String Formatting with %, Part 2

**Review**
16. And Now, For Something Completely Familiar

### Date and Time

**Date and Time**
1. The datetime Library
2. Getting the Current Date and Time
3. Extracting Information
4. Hot Date
5. Pretty Time
6. Grand Finale

## Unit 3 : Conditionals and Control Flow

### Conditionals & Control Flow

**Introduction to Control Flow**
1. Go With the Flow

**Comparators**
2. Compare Closely!
3. Compare... Closelier!
4. How the Tables Have Turned

**Boolean operators**
5. To Be and/or Not to Be
6. And
7. Or
8. Not
9. This and That(or This, But Not That!)
10. Mix 'n' Match

**If, Else and Elif**
11. Conditional Statement Syntax
12. If You're Having...
13. Else Problems, I Feel Bad for Y...
14. I got 99 Problems, But a Switc...

**Review**
15. The Big If

### PygLatin

**PygLatin Part 1**
1. Break It Down
2. Ahoy! (or Should I Say Ahoyay!)
3. Input!
4. Check Yourself!
5. Check Yourself... Some More
6. Pop Quiz!

**PygLatin Part 2**
7. Ay B C
8. Word Up
9. Move it on Back
10. Ending Up
11. Testing, Testing, is this Thing On?

Codecademy
# Language Skills
# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

Lesson : Python Syntax

This lesson will introduce you to Python, a general-purpose, object-oriented language you can use for countless stardalonone projects or scripting applications.

### Welcome!

Python is an easy to learn programming language.
You can use it to create web apps, games, even a search engine!

Ready to learn Python? Click Save & Submit Code to continue!

### Variables

Creating web apps, games, and search engines all involve storing and working with different types of data. They do so using variables. A variable stores a piece of data, and gives it a specific name.

for example :
```
spam = 5
```

The variable `spam` now stores the number `5.`

### Booleans

Great! You just stored a number in a variable. Numbers are one data type we use in programming. A second data type is called a boolean.

A boolean is like a light switch. It can only have two values. Just like a light switch can only be on or off, a boolean can only be `True` or `False.`

You can use variables to store booleans like this :
```
a = True
b = False
```

### You've Been Reassigned

Now you know how to use variables to store values.

Say `my_int = 7`. You can change the value of a variable by "reassigning" it, like this:
```
my_int = 3
```

### Whitespace

In Python, whitespace is used to structure code. Whitespace is important, so you have to be careful with how you use it.

### Whitespace Means Right Space

Now let's examine the error from the last lesson :
```
indentationError : expected an .....
```

You'll get this error whenever your whitespace is off.

### A Matter of Interpretation

The window in the top right corner of the page is called the interpreter.
The interpreter runs your code line by line, and check for any errors.

```
cats = 3
```

In the above example, we create a variable cats and assign it the value of 3.

### Single Line Comments

You probably saw us use the # sign a few times in earlier exercises. The # sign is a line of text that Python won't try to run as code. It's just for humans to read.

Comments make your program easier to understand. When you look back at your code or others want to collaborate with you, they can read your comments and easilly figure out what your code does.

### 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:

```
" " " sipping from your cup 'til....
Holy Grail.
" " "
```

### Math

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

```
addition = 72 + 23
subtraction = 108 - 204
multiplication = 100 * 0.5
division = 108 / 9
```

### 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

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.

### 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 `%`

### (+) Quiz

Which of the following is the exponent operator? : **
What is this variable equal to? : Variable = 3**3
What is this variable ez ...
"false" = string

Codecademy
# Language Skills
# Python
Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 1 : Python Syntax

Lesson : Tip calculator

Now that you've completed the lesson on Python syntax, let's see if you can put your newfound skills to use. In this lesson, you'll create a simple calculator that determines the price of a meal after tax and tip.

### The Meal

Now let's apply the concepts from the previous section to a real world example.
You've finished eating at a restauratn, and received this bill:

-Cost of meal : $44.50
-Restaurant tax : 6.75%
-Tip : 15%

You'll apply the tip to the overall cost of the meal (including tax).

### The Tax

Good! Now let's create a variable for the tax percentage.

The tax on your receipt is 6.75%. You'll have to divide 6.75 by 100 in order to get the decimal form of the percentage.
(See the Hint if you would like further explanation.)

### The Tip

Nice work! You received good service, so you'd like to leave a 15% tip on top of the cost of the meal, including tax.

Before we compute the tip for your bill, let's set a variable for the tip.
Again, we need to get the decimal form of the tip, so we divide 15.0 by 100.

### Reassign in a Single Line

Okay! We've got the three variables we need to perform our calculation, and we know some arithmetic operators that can help us out.

We saw in Lesson 1 that we can reassign variables. For example, we could say spam = 7,
then later change our minds and say spam = 3.

### The Total

Now that meal has the cost of the food plus tax, let's introduce on line 8 a new variable,
total, equal to the new meal + meal*tip.

The code on line 10 formats and prints to the console the value of total with exactly two numbers after the deimal.
(We'll learn about string formatting, the console, and print in Unit 2!)

Codecademy
# Language Skills
# Python
Learn to program in Python,
a powerful language used by sites like YouTube and Dropbox.

## Unit 2 : Strings and Console Output
Lesson : Strings & Console Output

Now that you've completed the lesson on Python syntax, let's see if you can put your newfound skills to use. In this lesson, you'll create a simple calculator that determines the price of a meal after tax and tip.

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

```
name = “Ryan”
age = “19”
food = “cheese”
```

01. In the above example, we create a variable name and set it to the string value “Ryan”.

02. We also set age to “19” and food to “cheese”.

Strings need to be within quotes.

### Practice
Excellent! Let's get a little practice in with strings.

### Excaping characters
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!’
```

### Access by Index
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]
```

01. 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”.

02. 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.

### 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:

01. len ( )
02. lower ( )
03. upper ( )
04. 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 ( )
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”.

### Dot Notation
Let's take a closer look at why you use len(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.

### Printing Strings
The area where we've been writing our code is called the editor.

The console ( the window in the upper right) is where the results of your code is shown.

print simply displays your code in the console.

### Printing Variables
Great! Now that we've printed strings, let's print variables.

### 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 Life and 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!

### 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 2 into a string and then you concatenate the strings together just like in the previous exercise.

Now try it yourself!

### String Formatting with %, Part 1
When you want to print a variable with a string, there is a better method than concatenating strings together.

```
name = “Mike”
print “Hello %s” % (name)
```

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.

```
ex)
string_1 = “camelot”
string_2 = “place”

print “Let's not go to %s. ‘Tis a silly %s.”
% (string_1, string_2)

> Let's not go to Camelot. ‘Tis a silly place.
```

### 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 %s terms 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!”
```

### 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)
```

## Unit 2 : Strings and Console Output

Lesson : Date and Time

This lesson is a follow up to Unit 2 : Strings and Console input and will give you practice with the concepts introduced in that lesson.

### 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 called datetime.now( ) to retrieve the current date and time.

```
from datetime import datetime
print datetime.now( )
```

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

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

### 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?

```
form 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 now and store it in current_year.

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

### 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 out 02/19/2014.

### 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.

### 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!

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 3 : Conditionals & Control Flow

Lesson : Conditionals & Control Flow

In this lesson, we'll learn how to create programs that generate different outcomes based on user input!

### 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 off what else is happening in the program.

### Compare Closely!

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

01. Equal to ( == )
02. Not equal to ( !=)
03. Less than ( < )
04. Less than or equal to ( <= )
05. Greater than ( > )
06. 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 tow things are equal, and = assigns a value to a variable.

### Compare... Closelier!

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

But what about extreme expressions and comparators?

### How the Tables have Turned

Comparisons result in either True or False, which are booleans as we learned before in this exercise.

```
#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.

### To Be and/ or Not to Be

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

01. and, which checks if both the statements are True ;
02. or, which checks if at least one of the statements is True ;
03. not, which gives the opposite of the statement.

We'll go through the operators one by one.

### And

The boolean operator and returns True when the expression on both sides of and are true. For instance :

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

### Or

The boolean operator or returns True when at least one expression on either side of or is true.
For example :

1 < 2 or 2 > 3 is True ;
1 > 2 or 2 > 3 is False.

### Not

The boolean operator not returns True for false statements and False for true statements.
For example :

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

01. not is evaluated first;
02. and is evaluated next;
03. or is evaluated last.

For example, True or not False and False 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.

### Mix 'n' Match

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

```
#Make me false
bool_one = (2 <= 2) and "Alpha" == "Bravo"
```

### Conditional Statement Syntax

if is a conditional statement that executes some specified code after checking if its expression is True.

Here's an example of if statement syntax :

```
if 8<9 :
   print "Eight is less than nine!"
```

In this example, 8<9 is the checked expression and print "Eight is less than nine!" is the specified code.

### If You're Having...

Let's get some practice with if statements. Remember, the syntax looks like this :

```
if some_function ( ) :
   # block line one
   # block line two
   # et cetera
```

Looking at the example above, in the event that some_funtion( ) 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 else don't get printed!"
```

### The Big If

Really great work! Here's what you've learned in this unit :

Comparators

```
3 < 4
5 >= 5
10 == 10
12 != 13
```

Boolean operators

```
True or False
(3<4) and (5>=5)
this( ) and not that ( )
```

Conditional statements

```
if this_might_be_true ( ) :
   print " This really is ture."
elif that_might_be_true ( ) :
   print " That is true."
else :
   print " None of the above."
```

Let's get to the grand finale.

Codecademy
# Language Skills
# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 3 : Conditionals & Control Flow

Lesson : PygLatin

In this lesson we'll put together all of the Python skills we've learned so far including string manipulation and branching. We'll be building a Pyg Latin translator. (That's Pig Latin for Python Programmers!)

### Break it Down

Now let's take what we've learned so far and write a Pig Latin translator.

Pig Latin is a language game, where you move the first letter of the word to the end and add "ay".
To write a Pig Latin translator in Python, here are the steps we'll need to take:

01. Ask the user to input a word in English.
02. Make sure the user entered in English
03. Convert the word from English to Pig Latin.
04. Display the translation result.

### Ahoy! (or Should I say Ahoyay!)

Let's warm up by printing a welcome message for our translator uses.

### Input!

Next, we need to ask the user for input.

```
name = raw_input("what's your name?")
print name
```

In the above example, raw_input( ) accepts a string, prints it, and then waits for the user to type something and press Enter ( or Return). In the Interpreter, Python will ask :

```
What's your name? >
```

Once you type in your name and hit Enter, if will be stored in name..

### Check Yourself... Some More

Now we know we have a non-empty sring. Let's be even more thorogh.

```
x = "J123"
x.isalpha  ( ) # False
```

In the first line, we create a string with letters and numbers.

The second line then runs the function isalpha ( ) which returns False since the string contains non-letter characters. Let's make sure the word the user enters contains only alphabetcal characters.
You can use isalpha( ) to check this! For Example :

### Pop quiz!

When you finish one part of your program, it's important to test it multiple times,
using a variety of inputs.

### Ay B C

Now we can get ready to start translating to Pig Latin! Let's review the rules for translation :
You move the first letter of the word to the end and then append the suffix 'ay'.
Example : python -> ythonpay

Let's create a variable to hold our translation suffix.

### Word Up

Let's simplify things by making he letters in our word lowercase.

```
the_string = "Hello"
the_string = the_string.lower()
```

The .lower( ) function does not modify the string itself, it simply returns a lowercase-version.
In the example above, we store the result back into the same variable.

We also need to grab the first letter of the word.

```
first_letter = the_string[0]
second_letter = the_string[1]
third_letter = the_string[2]
```

Remember that we start counting from zero, not one, so we access the firs letter by asking for [0].

### Move it on Back

Now that we have the first letter stored, we need to add both the letter and the string stored in pyg
to the end of the original string. Remember how to concatenate (i.e.add) strings together?

```
greeting = "Hello"
name = "D. Y."
welcome = greeting + name
```

### Ending Up

Well done! However, now we have the first leter showing up both at the beginning and near the end.

```
s = "Charlie"

print s[0]
#will print "C"
print s[1:4]
#will print "har"
```

01. first we create a variable s and give i the string "Charlie"
02. Next we access the first letter of "Charlie" using s[0]. Remember letter positions start at 0.
03. Then we access a slice of "Charlie" using s[1:4]. This returns everything from the letter
    at position 1 up till position 4.

We are going to slice the string just like in the 3rd example above.

### Testing, Testing, is This Thing On?

Yay! You should have a fully functioning Pig Latin translator. Test you code thorougly to be sure everything is working smoothly.

You'll also want to take out any print statements you were using to help debug intermediate steps of your code. Now might be a good time to add some comments too!
Making sure your code is clean, commented, and fully functional is just as important as writing it in the first place.

Codecademy
# Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 4 : Functions

---

### Introduction to Functions

**1. 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.

### Function Syntax

**2. Function Junction**

Functions are defined with three components:

01.
The header, which includes the def keyword, the name of the function,
and any parameters the function requrires. Here's an example:

```
def hello_world ( ) : // There are no parameters
```

02.
An optional comment that explains what the function does.

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

03.
The body, which describes the procedures the function carries out.
The body is indented, just like for conditional statements.

```
print  "Hello World!"
```

Here's the full function pieced togeter:

```
def hello_world():
   """Prints 'Hello World!" to the console."""
   print "Hello World!"
```

**3. 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.

**4. 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 square with 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.

**5. 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 exampl, a function can call another function :

```
def fun_one(n):
   return n*5

def fun_two(m):
   return fun_one(m) + 7
```

**6. Practice Makes Perfect**

Let's create a few more functions just for good measure.

```
def shout(phrase):
   if phrase == phrase.upper() :
      return "YOU'RE SHOUTING!"
   else :
      return "Can you speak up?"

shout("I'M INTERSETED IN SHOUTING!")
```

The example above is just there to help you remember how functions are structured.
Don't forget the colon at the end of your function definition!

### Importing Modules

**7. I know Kung Fu**

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

**8. Generic Imports**

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

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

**9. Function Imports**

Nice work! Now Python knows how to take the square root of a number.
However, we only really needed the sqrt function,
and it can be frustrating to have to keep typing math.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( )!

Codecademy
# Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 4 : Functions

Lesson : Functions

A function is a reusable section of code written to perform a specific task in a program. We gave you a taste of functions in Unit 3; here, you'll learn how to create your own.

### What Good are Functions?

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

### Function Junction

Functions are defined with three components:

01.
The header, which includes the def keyword, the name of the function, and any parameters the function requrires.
Here's an example:

```
def hello_world ( ) : // There are no parameters
```

02.
An optional comment that explains what the function does.

```
“””Prints  ‘Hello World!’ to the console.”””
```

03.
The body, which describes the procedures the function carries out.
The body is indented, just like for conditional statements.

```
          print  “Hello World!”
```

Here's the full function pieced togeter:

```
def hello_world():
          “””Prints ‘Hello World!” to the console.”””
          print “Hello World!”
```

### 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.

### 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 square with the argument 10.
In thie instance the function was calle, 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.

### 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 exampl, a function can call another function :

```
def fun_one(n):
  return n*5

def fun_two(m):
  return fun_one(m) + 7
```

### Practice Makes Perfect

Let's create a few more functions just for good measure.

```
def shout(phrase):
  if phrase == phrase.upper() :
    return “YOU’RE SHOUTING!”
  else :
    return “Can you speak up?”

shout(”I’M INTERSETED IN SHOUTING!”)
```

The example above is just there to help you remember how functions are structured.
Don't forget the colon at the end of your function definition!

### I Know Kung Fu

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

### Generic Imports

Did you see that? Python said : “NameError : name ‘sqrt’is not defined.”
Python doesn't know what squarer roots are--yet.

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

### Function Imports

Nice work! Now Python knows how to take the square root of a number.
However, we only really needed the sqrt function, and it can be frustrating to have to keep typing math.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 mor math.sqrt( )!

### 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 type math.?
Universal Import can handle this for you. The syntax for this is :

```
from module import *
```

### 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 your sqrt and there is math.sqrt. If you do from 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 type modul.name or just import specific variables and functions from various modules as neede.

### 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?

### mas( )

The max( ) function takes any number of arguments and returns the largest one.
(”Largest” can have odd definitions here, so it's best to use max( ) 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).

### min( )

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

### 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, 3 and -3 both have the same absolute value : 3. The abs( ) function always returns a positive value, and unlike mas( ) ans min( ), it only takes a single number.

### 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’>
```

### 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!

### Review : Modules

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

Hint) There are three ways you can import the sqrt( ) function, but we'd probably go with

```
from math import sqrt
```

You can figure out the rest. We believe in you!

### Review : Built-In Functions

Perfect! Last but not least, let's review the built-in functinos you've learned about in this lesson.

```
def is_numeric(num) :
  return type(num) == int or type(num) == float:
max(2,3,4) #4
min(2,3,4) #2
abs(2) #2
abs(-2) #2
```

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 4 : Functions

### ▶A Review of Function Creation

**1. 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:

01. We define a function called bigger that has two arguments called first and second.
02. Then, we print out the larger of the two arguments using the built-in function max.
03. Finally, the bigger function returns True.

Now Try creating a function yourself!

### ▶Planes, Hotels and Automobiles

**2. 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 you trip's costs.

**3. Getting There**
You're going to need to take a plane ride to get to you location.

```
def fruit_color(fruit):
   if fruit == “apple”:
      return “red”
   elif fruit == “banana”:
      return “yellow”
   elif fruit == “pear”:
      return “green”
```

01. The example above defiens the function fruit_color that accepts a string as the argument fruit.
02. The function returns a string if it knows the color of that fruit.

**4. 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.

01. First, we check if score is greater than or equal to 10. If it is, we give the player 50 bonus tickets.
02. If score is just greater than or equal to 7, we give the player 20 bonus tickets.
03. 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/ elif statements fail.

**5. 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)
```

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

**6. Hey, You Never Know!**
You can't expect to only spend money on the plane ride, hotel, and rental car when going ona vacation. There also needs to be room for additional costs like fancy food or souvenirs.

**7. 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?

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 5 : **Lists & Dictionaries**

## Python Lists and Dictionaries

---

### ▶Lists

#### 1. Introduction to Lists

Lists are a datatype you can use to store a collection of different pieces of information as a sequensce 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 strins, but there are a few key differences.

#### 2. 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.

#### 3. 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:

```
zoo_animals[2] = "hyena"
#Changes "sloth" to "hyena"
```

### ▶List Capabilities and Functions

#### 4. 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
```

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

#### 5. List Slicing

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

```
letters = ['a',  'b',  'c',  'd',  'e']
slice = letters[1:3]
print slice
print letters
```

01. In the above example, we first create a list called letters.
02. 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.
03. Next, we print out ['b', 'c']. Remember that we start counting indices from 0 and that we stopped before index 3.
04. Finally, we print out ['a', 'b', 'c', 'd', 'e'], just to show that we did not modify the original letters list.

#### 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.

#### 7. Maintaining Order

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

```
animals = ["ant",  "bat",  "cat"]
print animals.index("bat")
```

01. First, we create a list called animals with three strings.
02. 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
```

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

#### 8. For One and All

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

```
for variable in list_name:
   #Do stuff!
```

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

Then in list_name designates lit_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.

#### 9. 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
```

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

### ▶Dictionaries

#### 10. 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, key2':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!

#### 11. 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 lists inside dictionaries!)

#### 12. 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
```

#### 13. 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"]
```

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

#### 14. It's Dangerous to Go Alone! Take This

Let's go over a few last notes about dicionariesOO

```
my_dict = {
          "fish" : ["c", "a", "r", "p"]
          "cash" : -4483,
          "luck" : "good"
}
print my_dict["fish"][0]
```

01. In the example above, we created a dictionary that holds many types of values.
02. The key "fish" has a list, the key "cash" has an int, and the key "luck" has a string.
03. 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"

Codecademy
# Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 5 : **Lists & Dictionaries**

---

### ▶Looping with Lists and Dictionaries

#### 1. BeFOR We Begin
Before we begin our exercise, we should go gover 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 cool for 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 eing 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(currently item), 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.

#### 2. This is 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 throuth a dictionary, you will go through every key, but you are not guaranteed to get them in any particular order.

#### 3. 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."
```

01. In the above example, we create a list with 4 numbers in it.
02. Then we loop through the numbers list and store each item in the list in the variable number.
03. On each loop, if number is greater than 6, we print it out. So, we print 7.
04. Finally, we print out a sentence.

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

#### 4. 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
retrun total

lost = [4,8,15,16,23,42]
small = count_small(lost)
print small
```

01. In the above example, we define a function count_small that has one argument, numbers.
02. We initialize a variable total that we can use in the for loop.
03. For each item n in numbers, if n is less than 10, we increment total.
04. After the funcion definition, we create an array of numbers called lost.
05. After the function definition, we create an array of numbers called lost.
06. We call the count_small function, pass in lost, and store the returned result in small.
07. Finally, we print out the returned result, which is 2 since only 4 and 8 are less than 10.

#### 5. 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 throuth a string might work.

### ▶Owning a Store
#### 6. 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.

#### 7. Investing in Stock
Good work! As a store manager, you're also in charge of keeping track of your stock/inventory.

#### 8. 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]
```

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

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

### ▶Shopping Trip!
#### 10. 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 eye 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!

#### 11. 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)
```

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

#### 12. 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!

#### 13. Let's Check Out!
Perfect! You've done a great job with lists and dictionaries in this project. You've practiced:

-Using for loops with lists and dictionaries
-Writing functions with loops, lists, and dictionaries
-Updating data in response to change in the environment (for instance, decreasing the number of bananas in stock by 1 when you sell one).

Thanks for shopping at the Codecademy supermarket!

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 6 : Student Becomes the Teacher

### ▶ Good Morning Class!

**1. 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 isjust to remind you how to create a dictionary and then to access the item stored by the "cat" key.

**2. What's the Score?**

Great work!

**3. Put It Together**

Now lets put the three dictionaries in a list together.

```
my_list = [1,2,3]
```

The above example is just a reminder on how to create a list.
Afterwards, my_list contains 1, 2, and 3.

**4. For the Record**

Excellent. Now you need a hard copy document with all of your students' grades.

```
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.

**5. It's Okay to be Average**

When teaching a class, it's important to take the students' averages in order to assign grades.

```
5/2
# 2

5.0/2
#2.5

float(5) / 2
# 2.5
```

The above example is a reminder of how division works in Python.

01. When you divide an integer by another integer, the result is always an integer (rounded down, if needed).
02. When you divide a float by an integer, the result is always a float.
03. To divide two integers and end up with a float, you must first use float( ) to convert one of the integers to a float.

**6. Just Weight and See**

Great! Now we need to compute a student's average using weighted averages.

```
cost = {
   "apples" : [3.5, 2.4, 2.3],
   "bananas" : [1.2, 1.8],
}

return 0.9*average(cost["apples"])
0.1*average(cost["bananas"])
```

01. In the above example, we create a dictionary called cost that contains the costs of some fruit.
02. Then, we calculate the average cost of apples and the average cost of bananas. Since we like apples much more than we like bananas, we weight the average cost of apples by 90% and the average cost of bananas by 10%.

The \ character is a continuation character. The following line is considered a continuation of the current line.

**7. Sending a Letter**

Great work!
Now let's write a get_letter_grade function that takes a number score as input and returns a string with the letter grade that that student should receive.

**8. Part of the Whole**

Good! Now let's calculate the class average.
You need to get the average for each student and then calculate the average of those averages.

**9. How is Everybody Doing?**

Awesome! You're doing great.

# Python

Learn to program in Python,
a powerful language used by sites like YouTube and Dropbox.

## Unit 7 : Lists and Functions

### List Recap

**1. List accessing**
This exercise goes over just pulling information from a list, which we've covered in a previous section!

**2. List element modification**
You've already learned how to modify elements of a list in a previous section.
This exercise is just a recap of that!

**3. Appending to a list**
Here, we'll quickly recap how to .append( ) elements to the end of a list.

**4. Removing elements from lists**
This exercise will expand on ways to remove items from a list. You actually have a few options.
For a list called n :

01. n.pop(index) will remove the item at index from the list and return it to you:

```
n = [1,3,5]
n.pop(1)
#Return 3 (the item at index 1)
print n
#prints [1,5]
```

01. n.remove(item) will remove the actual item if it finds it:

```
n.remove(1)
#Removes 1 from the lists,
#NOT the item at index 1
print n
# prints [3.5]
```

01. del(n[1]) is like .pop in that it will remvoe the item at the given index, but it won't return it.

```
del(n[1])
#Doesn't return anything
print n
#prints [1,5]
```

### Function Recap

**5. Changing the functionality of a function**
In this exercise, you will just be making a minor change to a function to change what that function does.

**6. More than one argument**
This exercise is to recap how to use more than one argument in a function.

**7. Strings in functions**
This is a basic recap on using strings in functions.

### Introduction to Using Functions with Lists

**8. Passing a list to a function**
You pass a list to a function the same way you pass any other argument to a function.

**9. Using an element from a list in a function**
Passing a list to a function will store it in the argument (just like with a string or a number!)

```
def first_item(items) :
   print items[0]

numbers = [2,7,9]
first_item(numbers)
```

01. In the example above, we define a function called first_item. It has one argument called items.
02. Inside the function, we print out the item stored at index zero of items.
03. After the function, we create a new list called numbers.
04. Finally, we call the first_item function with numbers as its argument, which prints out 2.

**10. Modifying an element of a list in a function**
Modifying an element in a list in a function is the same as if you were just modifying an element of a list outside a function.

```
def double_first(n):
   n[0] = n[0]*2

numbers = [1,2,3,4]
double_first(numbers)
print numbers
```

01. We create a list called numbers.
02. We use the double_first function to modify that list.
03. Finally, we print out [2,2,3,4]

When we pass a list to a function and modify taht list, like in the double_first function above,
we end up modifyin the original list.

**11. List manipulation in functions**
You can also append or delete items of a list inside a function just as if you were manipulating the list outside a function.

```
my_list = [1,2,3]
my_list.append(4)
print my_list
# prints [1,2,3,4]
```

The example above is just a reminder of how to append items to a list.

### Using the Entire List in a Function

**12. Printing out a list item by item in function**
This exercise is to go over how to utilize every element in a list in a function. You can use the existing code to complete the exercise and see how running this operation inside a function isn't much different from running this operation outside a function.

Don't worry about the range function quite yet- we'll explain it later in this section.

**13. Modifying each element in a list in a function**
This exercise shows how to modify each element in a list. It is useful to do so in a fucntion as you can easily put in a list of any length and get the same functionality. As you can see, len(n) is the length of the list.

**14. Passing a range into a function**
Okay! Range time. The Python range( ) function is just a shortcut for generating a list, so you cna use ranges in all the same palces you can use lists.

```
range(6) # => [0,1,2,3,4,5]
range(1,6) # => [1,2,3,4,5]
range(1,6,3) # => [1,4]
```

The range function has three different versions:

01. range(stop)
02. range(start, stop)
03. range(start, stop, step)

In all cases, the range( ) function returns a list of numbers from start up to (but not including) stop.
Each item increases by step.

If omitted, start defaults to zero and step defaults to one.

**15. Iterating over a list in a function**
Now that we've learned about range, we have two ways of iterating through a list.

Method 1 - for item in list:

```
for item in list:
   print item
```

Method 2 - iterate through indexes:

```
for i in range(len(list)):
   print list[i]
```

Method 1 is useful to loop through the list, but it's not possible to modify the list this way.
Method 2 uses indexes to loop through the list, making it possible to also modify the list if needed.
Since we aren't modifying the list, feel free to use either one on this lesson!

**16. Using strings in lists in functions**
Now let's try working with strings!

```
for item in list:
   print item

for i in range(len(list)):
   print list[i]
```

The example above is just a reminder of the two methods for iterating over a list.

### Using Lists of Lists in Functions

**17. Using two lists as two arguments in a function**
Using multple lists in a function is no different from just using multiple arguments in a function!

```
a = [1,2,3]
b = [4,5,6]
print a+b
# prints [1,2,3,4,5,6]
```

The example above is just a reminder of how to concatenate two lists.

**18. Using a list of lists in a function**
Finally, this exercise shows how to make use of a single list that contains multiple lists and how to use them in a function.

```
list_of_lists = [[1,2,3], [4,5,6]]

for lst in list_of_lists:
   for item in lst:
      print item
```

01. In the example above, we first create a list containing two items, each of which is a list of numbers.
02. Then, we iterate through our outer list.
03. For each of the two inner lists (as lst), we iterate through the numbers (as item) and print them out.

We end up printing out:

```
1
2
3
4
5
6
```

Codecademy
Language Skills

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 7 : Lists and Functions

**Lists and Functions**

practice) **Battleship!**

**Lists and Functions**

---

### Don't Sink My Battleship!

**1. Welocome to Battleship!**
In this project you will build a simplified, one-player version of the classic board game Battleship!
In this version of the game, there will be a single ship hidden in a random location on a 5x5 grid.
The player will have 10 guesses to try to sink the ship.

To build this game we will use our knowledge of lists, conditionals and functions in Python.
When you're ready to get started, click run to continue.

**2. Getting Our Feet Wet**
The first thing we need to do is to wet up the game board.

**3. Make a List**
Good! now we'll use a built-in Python function to generate our board, which we'll make into a 5x5 grid of all "o"s, for "ocean".

```
print["o"]*5
```

will print out ['o', 'o', 'o', 'o', 'o'] which is the basis for a row of our board.

We'll do this five times to make five rows. (Since we have to do this five times, it sounds like a loop might be in order.)

**4. Check it Twice**
Great job! Now that we've built our board, let's show it off.

Throughout our game, we'll want to print the game board so that the player can see which locations they have already guessed. Regularly printing the board will also help us debug our program.

The easiest way to print the board would be to have Python display it for us using the print command.
Let's give that a try and see what the results look like- is this a useful way to print our board for Battleship?

**5. Custom Print**
Now we can see the contents of our list, but clearly it would be easier to play the game if we could print the board out like a grid with each row on its own line.

We can use the fact that our board is a list of lists to help us do this. Let's set up a for loop to go through each of the elements in the outer list(each of which is a row of our board) and print them.

**6. Printing Pretty**
We're getting pretty close to a playable board, but wouldn't it be nice to get rid of those quote marks and commas? We're storing our data as a list, but the player doesn't need to know that!

```
letters = ['a', 'b', 'c', 'd']
print " ".join(letters)
print "---".join(letters)
```

01. In the example above, we create a list called letters.
02. Then, we print a b c d. The .join method uses the string to combine the items in the list.
03. Finally, we print a---b---c---d. We are calling the .join function on the "---" string.

We want to trun each row into
" O O O O O "

**7. Hide...**
Excellent! Now, let's hide our battleship in a random location on the board.

Since we have a 2-dimensional list, we'll use two variables to store the ship's location, ship_row and ship_col.

```
from random import randint
coin = randint(0,1)
dice = randint(1,6)
```

01. In the above example, we first import the randint(low,high) function from the random module.
02. Then, we generate either zero or one and store it in coin.
03. Finally, we generate a number from one to six inclusive.

Let's generate a random_row and random_col from zero to four!

**8. ...and Seek!**
Good job! For now, let's store coordinates for the ship in the variables ship_row and ship_col.
Now you have a hidden battleship in your ocean!
Let's write the code to allow the player to guess where it is.

```
number = raw_input("Enter a number:  ")
if int(number) == 0:
   print "You entered 0"
```

raw_input asks the user for input and returns it as a string. But we're going to want to use integers for our guesses! To do this, we'll wrap the raw_inputs with int( ) to convert the string to an integer.

**9. It's Not Cheating - It's Debugging!**
Awesome! Now we have a hidden battleship and a guess from our player. In the next few steps, we'll check the user's guess to see if they are correct.

While we're writing and debugging this part of the program, it will be helpful to know where that battlehip is hidden. Let's add a print statement that displays the location of the hidden ship.

Of course, we'll remve this output when we're finished debugging snce if we left it in,
our game wouldn't be very challenging : )

**10. You win!**
Okay- now for the fun! We have the actual location of the ship and the player's guess so we can check to see if the player guessed right.

For a guess to be right, guess_col should be equal to ship_col and guess_row should be equal to ship_row.

```
if guess_col == 0 and guess_row == 0:
   print "Top-left corner."
```

The example above is just a reminder about if statements.

**11. Danger, Will Robinson!**
Great! Of course, the player isn't going to guess right all the time, so we also need to handle the case where the guess is wrong.

```
print board[2][3]
```

The example above prints out "0", the element in the 3rd row and 4th column.

**12. Bad Aim**
Great job! Now we can handle both correct and incorrect guesses from the user. But now let's think a little bit more about the "miss" condition.

01. They can enter a guess that's off the board.
02. They can guess a spot they've already guessed.
03. They can just miss the ship.

We'll add these tests inside our else condition. Let's build the first case now!

```
if x not in rane (8) or \
  y not in range(3):
       print "Outside the range"
```

The example above checks if either x or y are outside those ranges. The \ character just continues the if statement onto the next line.

**13. Not Again!**
Great! Now let's handle the second type of incorrect guess : the player guesses a location that was already guessed. How will we know that a location was previously guessed?

```
print board [guess_row] [guess_col]
```

The example above will print an 'x' if already guessed or an 'o' otherwise.

**14. Test Run**
Congratulations! Now you should have a game of Battleship! that is fully functional for one guess.
Make sure you play it a couple of times and try different kinds of incorrect guesses. This is a great time to stop and do some serious debugging.

In the next step, we'll move on and look at how to give the user 4 guesses to find the battleship

### You Sunk My Battleship!

**15. Play It, Sam**
You can successfully make one guess in Battleship! But we'd like our game to allow the player to make up to 4 guesses before they lose.

```
for turn in range(4):
   #Make a guess
   #Test that guess
```

We can use a far loop to iterate through a range. Each iteration will be a turn.

**16. Game Over**
If someone runs out of guesses without winning right now, the game just exits.
It would be nice to let them know why.

Since we only want this message to display if the user guesses wrong on their last turn, we need to think carefully about where to put it.

01. We'll want to put it under the else that accounts for misses
02. We'll want to print the message no matter what the cause of the miss
03. Since our turn variable starts at 0 and goes to 3, we will want to end the game when turn equals 3.

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 8 : **Loops**

### While Loops

**1. While You're here**

The while loop is similar to an if statement : it executes the code inside of it if some condition is true.
The difference is that the while loop will continue to execute as long as the condition is true.
In other words, instead of executing if something is true, it executes while that thing is true.

Line 6 decides when the loop will be executed. So, "as long as count is less than 5,"
the loop will continue to execute. Line8 increases count by 1. This happens over and over until count equals 5.

**2. Condition**

The condition is the expression that decides whether the loop is going to be executed or not.
There are 5 steps to this program:

01. The loop_condition variable is set to True.
02. The while loop checks to see if loop_condition is True. It is, so the loop is entered.
03. The print statement is executed.
04. The variable loop_condition is set to False
05. The while loop again checks to see if loop_conditions is True.
It is not, so the loop is not executed a second time.

**3. While you're at it**

Inside a while loop, you can do anything you could do elsewhere, including arithmetic operations.

**4. Simple errors**

A common application of a while loop is to check user input to see if it is vaild. For example, if you ask the user to enter y or n and they instead enter 7, then you should re-prompt them for input.

**5. Infinite loops**

An infinite loop is a loop that never exits. This can happen for a few reasons :

01. The loop condition cannot possibly be false (e.g. while 1 != 2)
02. The logic of the loop prevents the loop condition from becoming false.

Example :

```
count = 10
while count > 0:
   count += 1 # Instead of count
```

**6. Break**

The break is a one-line statement that means "exit the current loop." An alternate way to make our counting loop exit and stop executing is with the break statement.

-
First, create a while with a condition that is always true. The simplest way is shown.
-
Using an if statement, you define the stopping condition. Inside the if, you write break, meaning "exit the loop"

The difference here is that this loop is guaranteed to run at least once.

**7. While/ else**

Something completely different about Python is the while/else construction. while/else is similar to if/else, but there is a difference : the else block will execute anytime the loop conditionis evaluated to False.
This means that it will execute if the loop is never entered or if the loop exits normally. If the loop exits as the result of a break, the else will not be executed.

In this example, the loop will break if a 5 is generated, and the else will not execute.
Otherwise, after 3 numbers are generated, the loop condition will become flase and the else will execute.

**8. Your own while/ else**

Now you should be able to make a game similar to the one in the last exercise. The code from the last exercise is below :

```
count = 0
while count < 3:
   num = random.randint(1,6)
   print num
   if num == 5 :
      print "Sorry, you lose!"
      break
   count += 1
else:
   print "You win!"
```

In this exercise, allow the user to guess what the number is three times.

```
guess = int(raw_input("Your guess:  "))
```

Remember, raw_input turns user input into a string, so we use int( ) to make it a number again.

**9. For your health**

An alternative way to loop is the for loop. The syntax is as shown; this example means "for each number i in the range 0- 9, print i"

**10. For your hobbies**

This kind of loop is useful when you want to do something a certain number of times, such as append something to the end of a list.

**11. For your strings**

Using a for loop, you can print out each individual character in a string.
The example in the editor is almost plain English : "for each character c in thing, print c"

**12. For your A**

String manipulation is useful in for loops if you want to modify some content in a string.

```
word = "Marble"
for char in word :
   print char,
```

The example above iterates through ach character in word and, in the end, prints out  M a r b l e.

The, chaacter after our print statement means that our next print statement keeps printing on the same line.

**13. For your lists**

Perhaps the most useful (and most common) use of for loops is to go through a list.

On each iteration, the variable num will be the next value in the list. So, the first time through, it will be 7, the second time it will be 9, then 12,54,99, and then the loop will exit when there are no more values in the list.

**14. Looping over a dictionary**

You may be wondering how looping over a dictionary would work. Would you get the key or the value?

The short answer is : you get the key which you can use to get the value.

```
d = {'x':9, 'y':10, 'z':20}
for key in d :
   if d[key] == 10:
      print "This dictionary has the value 10!"
```

01. First, we create a dictionary with strings as the keys and numbers as the values.
02. Then, we iterate through the dictionary, each time storing the key in key.
03. Next, we check if that key's value is equal to 10.
04. Finally, we print this dictionary has the value 10!

**15. Counting as you go**

A weakness of using this for-each style of iteration is that you don't know the index of the thing you're looking at.
Generally this isn't an issue, but at times it is useful to know how far into the list you are.
Thankfully the built-in enumerate function helps with this.

enumerate works by supplying a corresponding index to each element in the list that you pass it.
Each time you go through the loop, index will be one greater, and item will be the next item in the sequence.
It's very similar to using a normal for loop with a list, except this gives us an easy way to count how many items we've seen so far.

**16. Multiple lists**

It's also common to need to iterate over two lists at once. This is where the built-in zip function comes in handy.

zip will create pairs of elements when passed two lists, and will stop at the end of the shorter list.
zip can handle three or more lists as well!

**17. For / else**

Just like with while, for loops may have an else associated with them.

In this case, the else statement is executed after the for, but only if the for ends normally- that is, not with a break. This code will break when it hits 'tomato', so the else block won't be executed.

**18. Change it up**

As mentioned, the else block won't run in this case, since break executes when it hits 'tomato'

**19. Create your own**

To wrap up this lesson, let's create our own for / else statement from scratch.

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 8 : Loops

**Loops**

practice) **Practice Makes Perfect**

---

### Fun with Numbers

**1. Practice! Practice! Practice!**

The best way to get good at anything is a lot of practice. This lesson is full of practice problems for you to work on. This section will contain minimal instructions to help you solve these problems; instead, this section will help you work on taking your programming skills and applying them to real life problems.

The more challenging programs will contain some helpful hints to nudge you in the right direction. If you feel as if you are completely lost, fee free to check out the Q&A section for help (the link is on the very bottom left of your screen).

**2. is_even**

All right! Let's get started.
Remember how an even number is a number that is divisible by 2?

**3. is_int**

An integer is just a number without a decimal part (for instance, -17, 0, and 42 are all integers, but 98.6 is not) For the purpose of this lesson, we'll also say that a number with a decimal part that is all 0s is also an integer, such as 7.0.

This means that, for this lesson, you can't just test the input to see if it's of type int.

If the difference between a number and that same number rounded down is greater than zero, what does that say about that particular number?

**4. digit_sum**

Awesome! Now let's try something a little trickier. Try summing the digits of a number.

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 10 : Advanced Topics in Python

**Advanced Topics in Python**

**Iteration Nation**
**List Comprehensions**
**List Slicing**
**Lambdas**

In this lesson, we'll cover some of the more complex aspects of Python, including iterating over data structures, list comprehensions, list slicing, and lambda expressions.

**Introduction to Bitwise Operators**

**Binary Representation**
**The Bitwise Operators**
**A Bit More Complicated**

Bitwise operations directly manipulate bits--patterns of 0s and 1s.
Though they can be tricky to learn at first,
their speed makes them a useful addition to any programmer's toolbox.

## Advanced Topics in Python

### Iterators for Dictionaries

Let's start with iterating over a dictionary. Recall that a dictionary is just a collection of keys and values.

```
d = {
    "Name" : "Guido",
    "Age" : 56,
    "BDFL" : True
}
print d.items ( )
# => [('BDFL', True), ('Age', 56), ('Name', 'Guido')]
```

Note that the items( ) function doesn't return key/value pairs in any specific order.
(For more on this, see the Hint.)

### Keys( ) and values( )

Whereas items( ) returns an array of tuples with each tuple consisting of a key/value pair from the dictionary:

-The keys( ) function returns an array of the dictionary's keys, and
-The values( ) function returns an array of the dictionary's values.

Again, these functions will not return the keys or values from the dictionary in any specific order.

### The 'in' Operator

For iterating over lists, tuples, dictionaries, and strings, Python also includes a special keyword: in.
You can use in very intuitively, like so :

```
for number in range(5):
    print number,

d = { "name": "Eric", "age": 26 }
for key in d:
    print key, d[key],

for letter in "Eric":
    print letter, # note the comma!
```

## List Comprehension

### Building Lists

Let's say you wanted to build a list of the numbers from 0 to 50 (inclusive). We could do this pretty easily :

```
my_list = range(51)
```

But what if we wanted to generate a list according to some logic-
for example, a list of all the even numbers from 0 to 50?

Python's answer to this is the list comprehension. List comprehensions are a powerful way to generate lists using the for/ in and if keywords we've learned.

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 11 : **Introduction to Classes**

---

### Class Basics

#### 1. Why Use Classes?

Python is an object-oriented programming language, which means it manipulates programmin constructs called objects. You can think of an object as a single data structure that contains data as well as functions; functions of objects are called mothods. For example, any time you call

```
len("Eric")
```

Python is checking to see whether the string object you passed it has a length, and if it does, it returns the value associated with that attribute. When you call

```
my_dict.items( )
```

Python checks to see if my_dict has an items( ) method (which all dictionaries have) and executes that method if it finds it.

But what makes "Eric" a string and my_dict a dictionary? The fact that they're instances of the str and dict classes, respectively.
A class is just a way of organizeing and producing objects with similar attributes and methods.

#### 2. Class Syntax

A basic class consits only of the class keyword, the name of the class, and the class from which the new class inherits in parentheses. (We'll get to inheritance soon.) For now, our classes will inherit from the object class, like so:

```
class NewClass(object):
    # Class magic here
```

This gives them the powers and abilities of a Python object. By convention, user-defined Python class names start with a capital letter.

#### 3. Classier Classes

We'd like our classes to do more than... well, nothing, so we'll have to replace our pass with something else.

You may have noticed in our example back in the first exercise that we started our class definition off with an odd- looking function : __init__( ).
This function is required for classes, and it's used to initialize the objects it creates.
__init__( ) always takes at least one argument, self, that refers to the object being created.
You can think of __init__( ) as the function that "boots up" each object the class creates.

#### 4. Let's Not Get too Selfish

Excellent! Let's make one more tweak to our class definition, then go ahead and instantiate (create) our first object.

So far, __init__( ) only takes one parameter : self. This is a Python convention; there's nothing magic about the word self. However, it's overshelmingly common to use self as the first parameter in __init__( ), so you should do this so that other people will understand your code.

The part that is magic is the fact that self is the first parameter passed to __init__( ). Python will use the first parameter that __init__( ) receives to refer to the object being created; this is why it's often called self, since this parameter gives the object being created its identity.

#### 5. Instantiating Your First Object

Perfect! Now we're ready to start creating objects.

We can access attributes of our objects using dot notation Here's how it works:

```
claas Square(object):
    def __init__(self):
        self.sides = 4

my_shape = Square( )
print my_shape.slides
```

01. First we create a class named Square with an attribute sides.
02. Outside the class definition, we create a new instance of Square named my_shape and access that attribute using my_shape.sides.

### Member Variables and Functions

#### 6. More on __init__() and self

Now that you're starting to understand how classes and objects work, it's worth delving a bit more into __init__( ) and self. They can be confusing!

As mentioned, you can think of __init__( ) as the method that "boots up" a class' instance object: the init bit is short for "initialize."

The first argument __init__( ) gets is used to refer to the instance object, and by convention, that argument is called self. If you add additional arguments- for instance, a name and age for your animal- setting each of those equal to self.name and self.age in the body of __init__( ) will make it so that when you creat an instance object of your Animal class, you need to give each instance a name and an age, and those will be associated with the particular instance you create.

#### 7. Class Scope

Another important aspect of Python classes is scope. The scope of a variable is the context in which it's visible to the program.

It may surprise you to learn that not all variables are accessible to all parts of a Python program at all times. When dealing with classes, you can have variables that ar available everywhere (global variables), variables that are only available to members of a certain class (member variables), and variables that are only available to particular instances of a class (instance variables).

The same goes for functions: some are available everywhere, some are only available to members of a certain class, and still others are only available to particular instance objects.

#### 8. A Methodical Approach

When a class has its own functions, those functions are called methods. You've already seen one such method: __init__( ). But you can also define your own methods!

#### 9. They're Multiplying!

A class can have any number of member variables. These are variables that are available to all members of a class.

```
hippo = Animal ("Jake", 12)
cat = Animal ("Boots", 3)
print hippo.is_alive
hippo.is_alive = False
print hippo.is_alive
print cat.is_alive
```

01. In the example above, we create two instances of an Animal.
02. Then we print out True, the default value stored in hippo's is_alive member variable.
03. Next, we set that to False and print it out to make sure.
04. Finally, we print out True, the value stored in cat's is_alive member variable.
    We only changed the variable in hippo, not in cat.

Let's add another member varaible to Animal.

#### 10. It's Not All Animals and Fruits

Classes like Animal and Fruit make it easy to understand the concepts of classes and instances, but you probably won't see many zebras or lemons in real-world programs.

However, classes and objects are often used to model real-world objects. The cod in the editor is a more realistic demonstration of the kind of classes and objects you might find in commercial software.
Here we have a basic ShoppingCart class for creating shopping cart objects for website customers; though basic, it's similar to what you'd see in a real program.

#### 11. Warning: Here Be Dragons

Inheritance is a tricky concept, so let's go through it step by step.

Inheritance is the process by shich one class take on the attributes and methods of another, and it's used to express an is-a relationship. For example, a Panda is a bear, so a Panda class ould inherit from a Bear class. However, a Toyota is not a Tractor, so it shouldn't inherit from the Tractor class (even if they have a lot of attributes and methods in common). Instead, both Toyota and Tractor could ultimately inherit from the same Vehicle class.

#### 12. Inheritance Syntax

In Python, inheritance works like this :

```
class DerivedClass (BaseClass):
    # code goes here
```

where DerivedClass is the new class you're making and BaseClass is the class from which that new class inherits.

#### 13. Override!

Sometimes you'll want one class that inherits from another to not only take on the methods and attributes of its parent, but to override one or more of them.

```
class Employee(object):
    def __init__(self, name):
        self.name = name
    def greet(self, other):
        print "Hello, %s" % other.name

class CEO(Employess)
    def greet (self, other):
        print "Get back to work, %s!" % other.name

ceo = CEO("Emily")
emp = Employee("Steve")
emp.greet(ceo)
#Hello, Emily
ceo.greet(emp)
#Get back to work, Steve!
```

Rather than have a separate greet_underling method for our CEO, we override (or re-create) the greet method on top of the base Employee.greet method.
This way, we don't need to know what type of Employee we have before we greet another Employee.

#### 14. This Look Like a Job For...

On the flip side, sometimes you'll be working with a derived class (or subclass) and realize that you've overwirtten a method or attibute defínced in that class' base class (also called a parent or superclass) that you actually need. Have no fear!
You can directly access the attibutes or methods of a superclass with Python's built-in super call.

The syntax looks like this:

```
class Derived(Base):
    def m(self):
        return super(Derived, self).m( )
```

Where m ( ) is a method from the base class

#### 15. Class Basics

First things first : let's create a class to work with

Codecademy
**Language Skills**

# Python

Learn to program in Python,
a powerful language ussed by sites like YouTube and Dropbox.

## Unit 12 : File Input and Output

### ▶ Introduction to File I/O

**1. See it to Believe it**

Until now, the Python code you've been writing comes from one source and only goes to one place : you type it in at the keyboard and its results are displayed in the console. But what if you want to read information from a file on your computer, and/or wirte that information to another file?

This process is called file I/O (the "I/O" stands for "Input/Output"), and Python has a number of built-in functions that handle this for you.

Check out the code in the editor to the right. Note that you now have an extra output.txt tab, which is just an empty text file. That's all about to change!

**2. The open( ) Function**

Let's walk through the process of writing to a file one step at a time.
The first code that you saw executed in the previous exercise was this:

```
f = open ("output.txt", "w")
```

This told Python to open output.txt in "w" mode ("W" stands for "write"). We storde the result of this operation in a file object, f.

Doing this opens the file in write-mode and prepares Python to send data into the file.
*this part should have added Hint! Warning!

**3. Writing**

Good work! Now it's time to write some data to our output.txt file.

We've added the list comprehension frm the first exercise to the code in the editor.
Our goal in this exercise will be to write each element of that list to output.txt (shown in a new tab above the editor) with each number on its own line.

We can write to a Python file like so:

```
my_file.write("Data to be written")
```

The write( ) function takes a string argument, so we'll need to do a few things here:

You must close the file. You do this simply by calling my_file.close() (we did this for you in the last exercise). If you don't close your file, Python won't write to it properly. From here on out, you gotta close your files!

**4. Reading**

Excellent! You're a pro.
Finally, we want to know how to read from our output.txt file.
As you might expect, we do this with the read() function, like so :

```
print my_file.read( )
```

### ▶ The Devil's in the Details

**5. Reading Between the Lines**

What if we want to read from a file line by line, rather than pulling the entire file in at once.
Thnakfully, Python includes a readlne( ) function that does exactly that.

If you open a file and call .readline( ) on the file object, you'll get the first line of the file; subsequent calls to .readline( ) will return successive lines.

**6. PSA : Buffering Data**

We keep telling you that you always need to close your files after you're done writing to them. Here's why! During the I/O process, data is buffered : this means that it is held in a temporary location before being written to the file.

Python doesn't flush the buffer - that is, write data to the file - until it's sure you're done writing.
One way to do this is to close the file. If you write to a file without closing, the data won't make it to the target file.

**7. The 'with' and 'as' keywords**

Programming is all about getting the computer to do the work. Is there a way to get Python to automatically close our files for us?

Of course there is. This is Python.

You may not know this, but file objects contain a special pair of built-in methods : __enter__( ) and __exit__( ) . The details aren't important, but what is important is that when a file object's __exit__( ) method is invoked, it automatically closes the file. How do we invoke this method? With with and as.

The syntax looks like this:

```
with open("file","mode") as variable:
   # Read or write to the file
```

**8. Try it Yourself**

It worked! Our Python program successfully wrote to text.txt.

**9. Case Closed?**

Finally, we'll want a way to test whether a file we've opened is closed. Sometimes we'll have a lot of file objects open, and if we're not careful, they won't all be closed. How can we test this?

```
f = open("bg.txt")
f.closed
# False
f.close()
f.closed
# True
```

Python file objects have a closed attribute which is True when the file is closed and False otherwise.

By checkin file_object.closed, we'll know whether our file is closed and can call close() on it if it's still open.