Python Crash Course

Resources for Python Crash Course (1st edition), from No Starch Press.

Resources for the second edition are here. I'd love to know what you think about Python Crash Course; please consider taking a brief survey. If you'd like to know when additional resources are available, you can sign up for email notifications here.

Solutions - Chapter 10

Back to solutions.

10-1: Learning Python

Open a blank file in your text editor and write a few lines summarizing what you've learned about Python so far. Start each line with the phrase *In Python you can…* Save the file as *learning_python.txt* in the same directory as your exercises fro mthis chapter. Write a program that reads the file and prints what you wrote three times. Print the contents once by reading in the entire file, once by looping over the file object, and once by storing the lines in a list and then working with them outside the `with` block.

learning_python.txt:

```
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

learning_python.py:

```
filename = 'learning_python.txt'

print("--- Reading in the entire file:")
with open(filename) as f:
    contents = f.read()
print(contents)

print("\n--- Looping over the lines:")
with open(filename) as f:
    for line in f:
        print(line.rstrip())

print("\n--- Storing the lines in a list:")
with open(filename) as f:
    lines = f.readlines()

for line in lines:
    print(line.rstrip())
```

Output:

```
--- Reading in the entire file:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Looping over the lines:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Storing the lines in a list:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

top

10-2: Learning C

You can use the `replace()` method to replace any word in a string with a different word. Here's a quick example showing how to replace `'dog'` with `'cat'` in a sentence:

```
>>> message = "I really like dogs."
>>> message.replace('dog', 'cat')
'I really like cats.'
```

Read in each line from the file you just created, *learning_python.txt*, and replace the word *Python* with the name of another language, such as *C*. Print each modified line to the screen.

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # Get rid of newline, then replace Python with C.
    line = line.rstrip()
    print(line.replace('Python', 'C'))
```

Output:

```
In C you can store as much information as you want.
In C you can connect pieces of information.
In C you can model real-world situations.
```

You can use `rstrip()` and `replace()` on the same line. This is called *chaining* methods. In the following code the newline is stripped from the end of the line and then *Python* is replaced by *C*. The output is identical to the code shown above.

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # Get rid of newline, then replace Python with C.
    print(line.rstrip().replace('Python', 'C'))
```

top

10-3: Guest

Write a program that prompts the user for their name. When they respond, write their name to a file called *guest.txt*.

```
name = input("What's your name? ")

filename = 'guest.txt'

with open(filename, 'w') as f:
    f.write(name)
```

Output:

```
What's your name? eric
```

guest.txt:

```
eric
```

top

10-4: Guest Book

Write a `while` loop that prompts users for their name. When they enter their name, print a greeting to the screen and add a line recording their visit in a file called *guest_book.txt*. Make sure each entry appears on a new line in the file.

```
filename = 'guest_book.txt'

print("Enter 'quit' when you are finished.")
while True:
    name = input("\nWhat's your name? ")
    if name == 'quit':
        break
    else:
        with open(filename, 'a') as f:
            f.write(name + "\n")
        print("Hi " + name + ", you've been added to the guest book.")
```

Output:

```
Enter 'quit' when you are finished.

What's your name? eric
Hi eric, you've been added to the guest book.

What's your name? willie
Hi willie, you've been added to the guest book.

What's your name? ever
Hi ever, you've been added to the guest book.

What's your name? erin
Hi erin, you've been added to the guest book.

What's your name? quit
```

guest_book.txt:

```
eric
willie
ever
erin
```

top

10-5: Programming Poll

Write a `while` loop that asks people why they like programming. Each time someone enters a reason, add their reason to a file that stores all the responses.

```
filename = 'programming_poll.txt'

responses = []
while True:
    response = input("\nWhy do you like programming? ")
    responses.append(response)

    continue_poll = input("Would you like to let someone else respond? (y/n) ")
    if continue_poll != 'y':
        break

with open(filename, 'a') as f:
    for response in responses:
        f.write(response + "\n")
```

Output:

```
Why do you like programming? Programmers can build almost anything they can imagine.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It's really fun, and really satisfying.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It just never gets old.
Would you like to let someone else respond? (y/n) n
```

programming_poll.txt:

```
Programmers can build almost anything they can imagine.
It's really fun, and really satisfying.
It just never gets old.
```

top

10-6: Addition

One common problem when prompting for numerical input occurs when people provide text instead of numbers. When you try to convert the input to an `int`, you'll get a `ValueError`. Write a program that prompts for two numbers. Add them together and print the result. Catch the `TypeError` if either input value is not a number, and print a friendly error message. Test your program by entering two numbers and then by entering some text instead of a number.

```
try:
    x = input("Give me a number: ")
    x = int(x)

    y = input("Give me another number: ")
    y = int(y)

except ValueError:
    print("Sorry, I really needed a number.")

else:
    sum = x + y
    print("The sum of " + str(x) + " and " + str(y) + " is " + str(sum) + ".")
```

Output with two integers:

```
Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.
```

Output with non-numerical input:

```
Give me a number: 23
Give me another number: fred
Sorry, I really needed a number.
```

top

10-7: Addition Calculator

Wrap your code from Exercise 10-6 in a `while` loop so the user can continue entering numbers even if they make a mistake and enter text instead of a number.

```
print("Enter 'q' at any time to quit.\n")

while True:
    try:
```

```
        x = input("\nGive me a number: ")
        if x == 'q':
            break

        x = int(x)

        y = input("Give me another number: ")
        if y == 'q':
            break

        y = int(y)

    except ValueError:
        print("Sorry, I really needed a number.")

    else:
        sum = x + y
        print("The sum of " + str(x) + " and " + str(y) + " is " + str(sum) + ".")
```

Output:

```
Enter 'q' at any time to quit.

Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.

Give me a number: three
Sorry, I really needed a number.

Give me a number: 3
Give me another number: five
Sorry, I really needed a number.

Give me a number: -12
Give me another number: 20
The sum of -12 and 20 is 8.

Give me a number: q
```

top

10-8: Cats and Dogs

Make two files, `cats.txt` and `dogs.txt`. Store at least three names of cats in the first file and three names of dogs in the second file. Write a program that tries to read these files and print the contents of the file to the screen. Wrap your code ina `try-except` block to catch the `FileNotFound` error, and

print a friendly message if a file is missing. Move one of the files to a different location on your system, and make sure the code in the `except` block executes properly.

cats.txt:

```
henry
clarence
mildred
```

dogs.txt:

```
willie
annahootz
summit
```

cats_and_dogs.py:

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:
    print("\nReading file: " + filename)
    try:
        with open(filename) as f:
            contents = f.read()
            print(contents)
    except FileNotFoundError:
        print("  Sorry, I can't find that file.")
```

Output with both files:

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

Output after moving *cats.txt:*

```
Reading file: cats.txt
  Sorry, I can't find that file.

Reading file: dogs.txt
willie
```

```
annahootz
summit
```

top

10-9: Silent Cats and Dogs

Modify your `except` block in Exercise 10-8 to fail silently if either file is missing.

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:

    try:
        with open(filename) as f:
            contents = f.read()

    except FileNotFoundError:
        pass

    else:
        print("\nReading file: " + filename)
        print(contents)
```

Output when both files exist:

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

Output when *cats.txt* has been moved:

```
Reading file: dogs.txt
willie
annahootz
summit
```

top

10-11: Favorite Number

Write a program that prompts for the user's favorite number. Use `json.dump()` to store this number in a file. Write a separate program that reads in this value and prints the message, "I know your favorite number! It's _____."

favorite_number_write.py:

```
import json

number = input("What's your favorite number? ")

with open('favorite_number.json', 'w') as f:
    json.dump(number, f)
    print("Thanks! I'll remember that.")
```

Output:

```
What's your favorite number? 42
Thanks! I'll remember that.
```

favorite_number_read.py:

```
import json

with open('favorite_number.json') as f:
    number = json.load(f)

print("I know your favorite number! It's " + str(number) + ".")
```

Output:

```
I know your favorite number! It's 42.
```

top

10-12: Favorite Number Remembered

Combine the two programs from Exercise 10-11 into one file. If the number is already stored, report the favorite number to the user. If not, prompt for the user's favorite number and store it in a file. Run the program twice to see that it works.

```
import json

try:
    with open('favorite_number.json') as f:
```

```
        number = json.load(f)
except FileNotFoundError:
    number = input("What's your favorite number? ")
    with open('favorite_number.json', 'w') as f:
        json.dump(number, f)
    print("Thanks, I'll remember that.")
else:
    print("I know your favorite number! It's " + str(number) + ".")
```

Output, first run:

```
What's your favorite number? 42
Thanks, I'll remember that.
```

Output, second run:

```
I know your favorite number! It's 42.
```

top

10-13: Verify User

The final listing for *remember_me.py* assumes either that the user has already entered their username or that the program is running for the first time. We should modify it in case the current user is not the person who last used the program.

Before printing a welcome back message in `greet_user()`, ask the user if this is the correct username. If it's not, call `get_new_username()` to get the correct username.

```
import json

def get_stored_username():
    """Get stored username if available."""
    filename = 'username.json'
    try:
        with open(filename) as f_obj:
            username = json.load(f_obj)
    except FileNotFoundError:
        return None
    else:
        return username

def get_new_username():
    """Prompt for a new username."""
    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f_obj:
```

```
        json.dump(username, f_obj)
    return username

def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        correct = input("Are you " + username + "? (y/n) ")
        if correct == 'y':
            print("Welcome back, " + username + "!")
        else:
            username = get_new_username()
            print("We'll remember you when you come back, " + username + "!")
    else:
        username = get_new_username()
        print("We'll remember you when you come back, " + username + "!")

greet_user()
```

Output:

```
> python verify_user.py
What is your name? eric
We'll remember you when you come back, eric!

> python verify_user.py
Are you eric? (y/n) y
Welcome back, eric!

> python verify_user.py
Are you eric? (y/n) n
What is your name? ever
We'll remember you when you come back, ever!

> python verify_user.py
Are you ever? (y/n) y
Welcome back, ever!
```

You might notice the identical `else` blocks in this version of `greet_user()`. One way to clean this function up is to use an empty `return` statement. An empty `return` statement tells Python to leave the function without running any more code in the function.

Here's a cleaner version of `greet_user()`:

```
def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        correct = input("Are you " + username + "? (y/n) ")
```

```
        if correct == 'y':
            print("Welcome back, " + username + "!")
            return

    # We got a username, but it's not correct.
    # Let's prompt for a new username.
    username = get_new_username()
    print("We'll remember you when you come back, " + username + "!")
```

The `return` statement means the code in the function stops running after printing the welcome back message. When the username doesn't exist, or the username is incorrect, the `return` statement is never reached. The second part of the function will only run when the `if` statements fail, so we don't need an `else` block. Now the function prompts for a new username when either `if` statement fails.

The only thing left to address is the nested `if` statements. This can be cleaned up by moving the code that checks whether the username is correct to a separate function. If you're enjoying this exercise, you might try making a new function called `check_username()` and see if you can remove the nested `if` statement from `greet_user()`.

top