Lecture 6

Welcome!

- In previous weeks, you were introduced to the fundamental building blocks of programming.
- You learned about programming in a lower-level programming language called C.
- Today, we are going to work with a higher-level programming language called *Python*.
- As you learn this new language, you're going to find that you are going to be more able to teach yourself new programming languages.

Python

- Humans, over the decades, have seen how previous design decisions could be improved upon.
- Python is a programming language that builds upon what you have already learned in C.

Hello

- Up until this point, the code has looked like this:

```c
// A program that says hello to the world

#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

- Today, you'll find that the process of writing and compiling code has been simplified.
- For example, the above code will be rendered in Python as:

```python
# A program that says hello to the world

print("hello, world")
```

  Notice that the semicolon is gone.

- In C, you might remember this code:

```c
// get_string and printf with %s

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    string answer = get_string("What's your name? ");
    printf("hello, %s\n", answer);
}
```

- This code is transformed in Python to:

```python
# get_string and print, with concatenation

from cs50 import get_string
```

```
answer = get_string("What's your name? ")
print("hello, " + answer)
```

You can write this code by executing `code hello.py` in the terminal window. Then, you can execute this code by running `python hello.py`. Notice how the `+` sign concatenates `"hello, "` and `answer`.

- Similarly, you could implement the above code as:

```
# get_string and print, with format strings

from cs50 import get_string

answer = get_string("What's your name? ")
print(f"hello, {answer}")
```

Notice how the curly braces allow for the `print` function to interpolate the `answer` such that `answer` appears within.

Types

- Data types in Python do not need to be explicitly declared. For example, you saw how `answer` above is a string, but we did not have to tell the interpreter this was the case: It knew on its own.
- In Python, commonly used types include:

```
bool
float
int
str
```

Notice that `long` and `double` are missing. Python will handle what data type should be used for larger and smaller numbers.

- Some other data types in Python include:

```
range
list
tuple
dict
set
```

- Each of these data types can be implemented in C, but in Python they can be implemented more simply.

Speller

- To illustrate this simplicity, let's type 'code dictionary.py' in the terminal window and write code as follows:

```
# Words in dictionary
words = set()


def check(word):
    """Return true if word is in dictionary else false"""
    if word.lower() in words:
        return True
    else:
        return False


def load(dictionary):
    """Load dictionary into memory, returning true if successful else false"""
    file = open(dictionary, "r")
    for line in file:
        word = line.rstrip()
        words.add(word)
    file.close()
    return True


def size():
    """Returns number of words in dictionary if loaded else 0 if not yet loaded""
    return len(words)


def unload():
    """Unloads dictionary from memory, returning true if successful else false"""
    return True
```

  Notice that there are four functions above. In the `check` function, if a `word` is in `words`, it returns `True`. So much easier than an implementation in C! Similarly, in the `load` function the dictionary file is opened. For each line in that file, we add that line to `words`. Using `rstrip`, the trailing new line is removed from the added word. `size` simply returns the `len` or length of `words`. `unload` only needs to return `True` because Python handles memory management on its own.

- The above code illustrates why higher-level languages exist: To simplify and allow you to write code more easily.
- However, speed is a tradeoff. Because C allows you, the programmer, to make decisions about memory management, it may run faster than Python – depending on your code. While C only runs your lines of code, Python runs all the code that comes under the hood with it when you call Python's built-in functions.
- You can learn more about functions in the Python documentation (https://docs.python.org/3/library/functions.html)

Image Recognition

- Numerous libraries have been written by contributors to Python.
- You can utilize these libraries in your own code.
- For example, you could simply import facial recognition utilizing a Python library like `PIL`.
- David provided a demo of facial recognition utilizing Python and third-party libraries.

CS50 Library

- As with C, the CS50 library can be utilized within Python.
- The following functions will be of particular use:

```
get_float
get_int
get_string
```

- You also have the option of importing only specific functions from the CS50 library as follows:

```
from CS50 import get_float, get_int, get_string
```

Conditionals

- In C, you might remember a program like this:

```
// Conditionals, Boolean expressions, relational operators

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Prompt user for integers
    int x = get_int("What's x? ");
    int y = get_int("What's y? ");

    // Compare integers
    if (x < y)
    {
        printf("x is less than y\n");
    }
    else if (x > y)
    {
        printf("x is greater than y\n");
    }
```

```
    else
    {
        printf("x is equal to y\n");
    }
}
```

- In Python, it would appear as follows:

```
# Conditionals, Boolean expressions, relational operators

from cs50 import get_int

# Prompt user for integers
x = get_int("What's x? ")
y = get_int("What's y? ")

# Compare integers
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

Notice that there are no more curly braces. Instead, indentations are utilized. Second, a colon is utilized in the `if` statement. Further, `elif` replaces `else if`. Parentheses are also no longer required in the `if` and `elif` statements.

Variables

- Variable declaration is simplified too. In C, you might have `int counter = 1;`. In Python, this same line would read `counter = 1`. You need not declare the type of the variable.
- Python favors `counter += 1` to increment by one, losing the ability found in C to type `counter++`.

Loops

- Loops in Python are very similar to C. You may recall the following code in C:

```
// Demonstrates while loop

#include <stdio.h>

int main(void)
{
    int i = 0;
    while (i < 3)
    {
```

```
        printf("meow\n");
        i++;
    }
}
```

- In Python, this code appears as:

```
# Demonstrates while loop

i = 0
while i < 3:
    print("meow")
    i += 1
```

- `for` loops can be implemented in Python as follows:

```
# Better design

for i in range(3):
    print("meow")
```

- Similarly, one could express the above code as:

```
# Abstraction with parameterization

def main():
    meow(3)


# Meow some number of times
def meow(n):
    for i in range(n):
        print("meow")


main()
```

Notice that a function is utilized to abstract away the meowing.

Calculator

- We can implement a simple calculator just as we did within C. Type `code calculator.py` into the terminal window and write code as follows:

```
# Addition with int [using get_int]

from cs50 import get_int

# Prompt user for x
x = get_int("x: ")

# Prompt user for y
```

```
y = get_int("y: ")

# Perform addition
print(x + y)
```

Notice how the CS50 library is imported. Then, `x` and `y` are gathered from the user. Finally, the result is printed. Notice that the `main` function that would have been seen in a C program is gone entirely! While one could utilize a `main` function, it is not required.

- It's possible for one to remove the training wheels of the CS50 library. Modify your code as follows:

```
# Addition with int [using input]

# Prompt user for x
x = input("x: ")

# Prompt user for y
y = input("y: ")

# Perform addition
print(x + y)
```

Notice how executing the above code results in strange program behavior. Why might this be so?

- You may have guessed that the interpreter understood `x` and `y` to be strings. You can fix your code by employing the `int` function as follows:

```
# Addition with int [using input]

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Perform addition
print(x + y)
```

Notice how the input for `x` and `y` is passed to the `int` function which converts it to an integer.

- We can expand the abilities of our calculator. Modify your code as follows:

```
# Division with integers, demonstration lack of truncation

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))
```

```
# Divide x by y
z = x / y
print(z)
```

Notice that executing this code results in a value, but that if you were to see more digits after `.333333` you'd see that we are faced with *floating-point imprecision*.

- We can reveal this imprecision by modifying our codes slightly:

```
# Floating-point imprecision

# Prompt user for x
x = int(input("x: "))

# Prompt user for y
y = int(input("y: "))

# Divide x by y
z = x / y
print(f"{z:.50f}")
```

Notice that this code reveals the imprecision. Python still faces this issue, just as C does.

Compare

- In C, we faced challenges when we wanted to compare two values. Consider the following code:

```
// Conditionals, Boolean expressions, relational operators

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Prompt user for integers
    int x = get_int("What's x? ");
    int y = get_int("What's y? ");

    // Compare integers
    if (x < y)
    {
        printf("x is less than y\n");
    }
    else if (x > y)
    {
        printf("x is greater than y\n");
    }
    else
    {
        printf("x is equal to y\n");
```

```
    }
}
```

- In Python, we can execute the above as follows:

```
# Conditionals, Boolean expressions, relational operators

from cs50 import get_int

# Prompt user for integers
x = get_int("What's x? ")
y = get_int("What's y? ")

# Compare integers
if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x is equal to y")
```

Notice that the CS50 library is imported. Further, minor changes exist in the `if` statement.

- Further looking at comparisons, consider the following code in C:

```
// Logical operators

#include <cs50.h>
#include <stdio.h>

int main(void)
{
    // Prompt user to agree
    char c = get_char("Do you agree? ");

    // Check whether agreed
    if (c == 'Y' || c == 'y')
    {
        printf("Agreed.\n");
    }
    else if (c == 'N' || c == 'n')
    {
        printf("Not agreed.\n");
    }
}
```

- The above can be implemented as follows:

```
# Logical operators

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")
```

```
# Check whether agreed
if s == "Y" or s == "y":
    print("Agreed.")
elif s == "N" or s == "n":
    print("Not agreed.")
```

Notice that the two vertical bars utilized in C is replaced with `or`. Indeed, people often enjoy Python because it is more readable by humans. Also, notice that `char` does not exist in Python. Instead, `str`s are utilized.

- Another approach to this same code could be as follows:

```
# Logical operators, using lists

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")

# Check whether agreed
if s in ["y", "yes"]:
    print("Agreed.")
elif s in ["n", "no"]:
    print("Not agreed.")
```

Notice how we are able to express multiple keywords like `y` and `yes`.

Object-Oriented Programming

- Up until this point, our programs in this course have been linear: sequential.
- It's possible to have certain types of values not only have properties or attributes inside of them but have functions as well. In Python, these values are known as *objects*
- In C, we could create a `struct` where you could associate multiple variables inside a single self-created data type. In Python, we can do this and also include functions in a self-created data type. When a function belongs to a specific *object*, it is known as a *method*.
- For example, `strs` in Python have a built-in *methods*. Therefore, you could modify your code as follows:

```
# Logical operators, using lists

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")

# Check whether agreed
if s.lower() in ["y", "yes"]:
    print("Agreed.")
```

```
elif s.lower() in ["n", "no"]:
    print("Not agreed.")
```

Notice how we are able to express multiple keywords like `y` and `yes` and convert any user input to lowercase.

- This could be further simplified as:

```
# Logical operators, using lists

from cs50 import get_string

# Prompt user to agree
s = get_string("Do you agree? ")

s = s.lower()

# Check whether agreed
if s in ["y", "yes"]:
    print("Agreed.")
elif s in ["n", "no"]:
    print("Not agreed.")
```

Notice how the old value of `s` is overwritten with the result of `s.lower()`.

- In this class, we will only scratch the surface of Python. Therefore, the Python documentation (https://docs.python.org) will be of particular importance as you continue.
- You can learn more about string methods in the Python documentation (https://docs.python.org/3/library/stdtypes.html#string-methods)

Meow

- Returning to `meow.c` from weeks earlier, recall the following code:

```
// Demonstrates while loop

#include <stdio.h>

int main(void)
{
    int i = 0;
    while (i < 3)
    {
        printf("meow\n");
        i++;
    }
}
```

- The above can be implemented within Python as:

```
# Demonstrates while loop

i = 0
while i < 3:
    print("meow")
    i += 1
```

- Similarly, using a `for` loop, we can write code as follows:

```
# Better design

for i in range(3):
    print("meow")
```

- As we hinted at earlier today, you can further improve upon this code using functions. Modify your code as follows:

```
# Abstraction

def main():
    for i in range(3):
        meow()

# Meow once
def meow():
    print("meow")


main()
```

Notice that the `meow` function abstracts away the `print` statement. Further, notice that the `main` function appears at the top of the file. At the bottom of the file, the `main` function is called. By convention, it's expected that you create a `main` function in Python.

- Indeed, we can pass variables between our functions as follows:

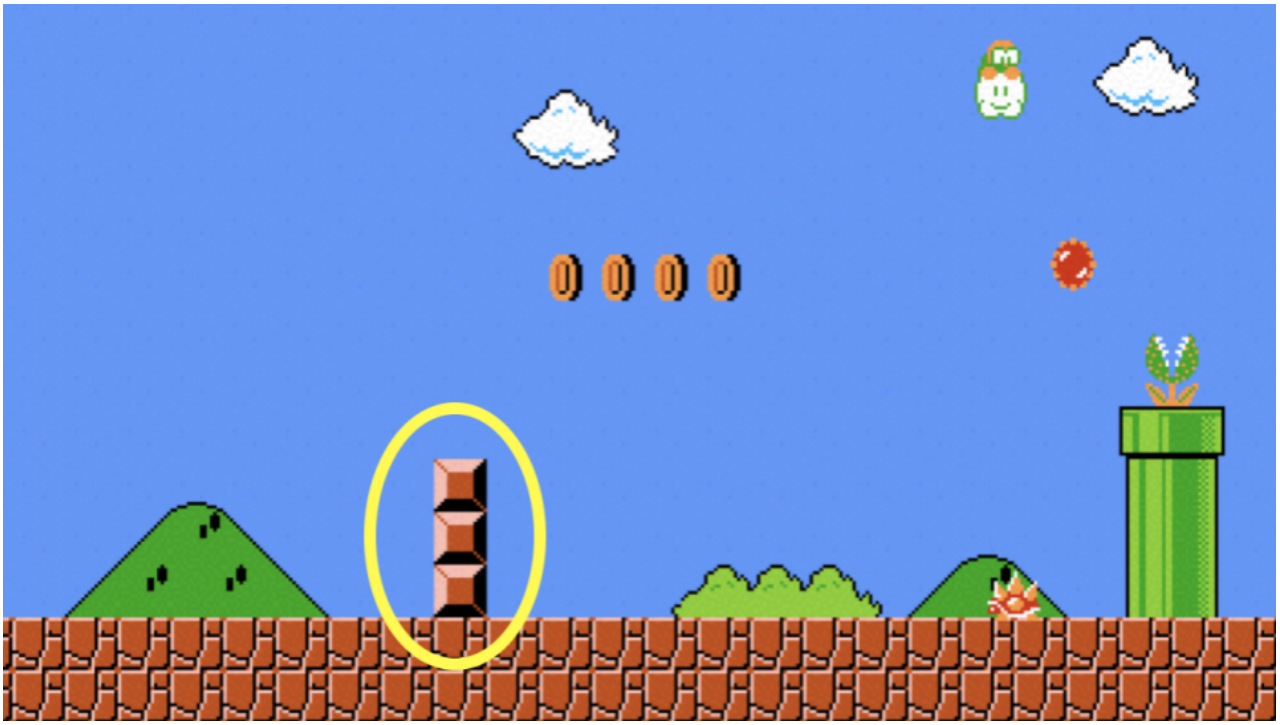
```
# Abstraction with parameterization

def main():
    meow(3)


# Meow some number of times
def meow(n):
    for i in range(n):
        print("meow")


main()
```

Notice how `meow` now takes a variable `n`. In the `main` function, you can call `meow` and pass a value like `3` to it. Then, `meow` utilizes the value of `n` in the `for` loop.

- Reading the above code, notice how you, as a C programmer, are able to quite easily make sense of the above code. While some conventions are different, the building blocks you previously learned are very apparent in this new programming language.

Mario

- Recall a few weeks ago our challenge of building three blocks on top of one another, like in Mario.



- In Python, we can implement something akin to this as follows:

```
# Prints a column of 3 bricks with a loop

for i in range(3):
    print("#")
```

- In C, we had the advantage of a `do-while` loop. However, in Python it is convention to utilize a `while` loop, as Python does not have a `do while` loop. You can write code as follows in a file called `mario.py`:

```
# Prints a column of bricks, using a helper function to get input

from cs50 import get_int


def main():
    height = get_height()
    for i in range(height):
        print("#")
```

```
def get_height():
    while True:
        n = get_int("Height: ")
        if n > 0:
            return n


main()
```

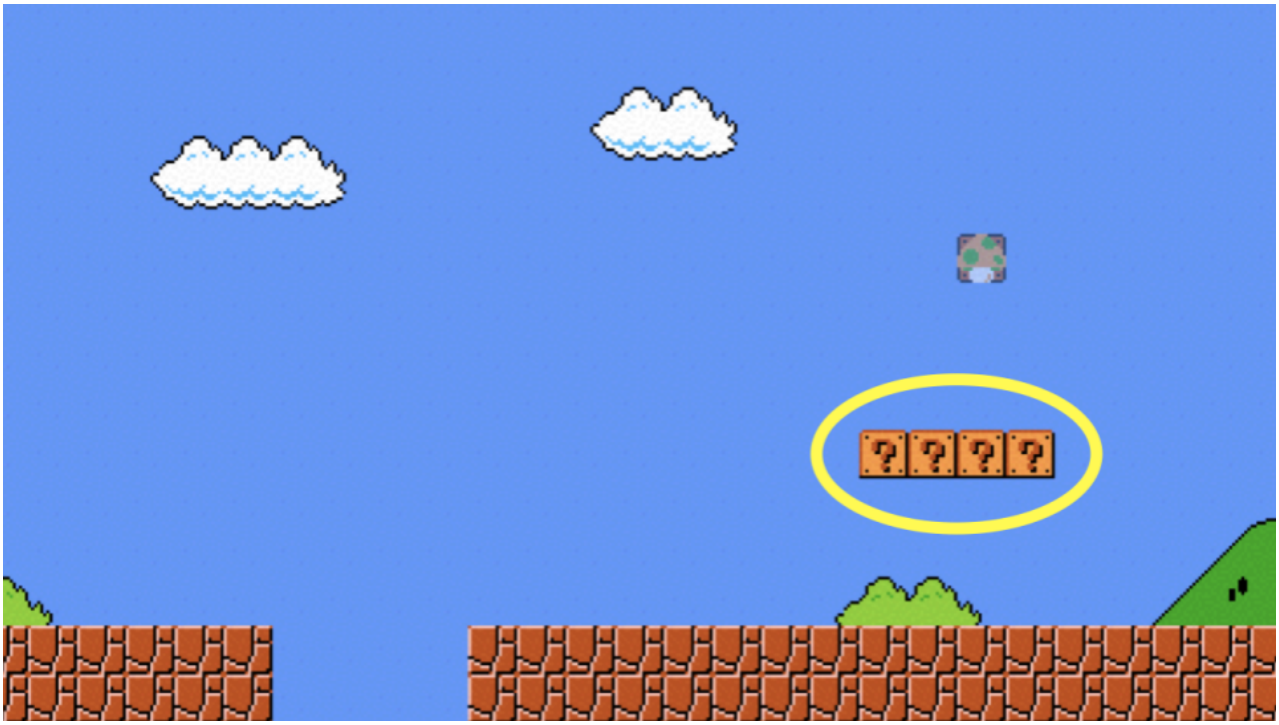
Notice how the scope of `n` is everywhere in the `get_height` function once it is assigned a value. Further notice that by convention, there are double spaces between functions.

- We can take away the training wheels of the CS50 library as follows:

```
# Prints a column of bricks, catching exceptions

def main():
    height = get_height()
    for i in range(height):
        print("#")


def get_height():
    while True:
        try:
            n = int(input("Height: "))
            if n > 0:
                return n
        except ValueError:
            print("Not an integer")


main()
```

Notice that `try` is utilized to attempt to convert `n` to an integer. If it cannot do so, an error is outputted.

- Consider the following image:



- In Python, we could implement by modifying your code as follows:

```
# Prints a row of 4 question marks with a loop

for i in range(4):
    print("?", end="")
print()
```

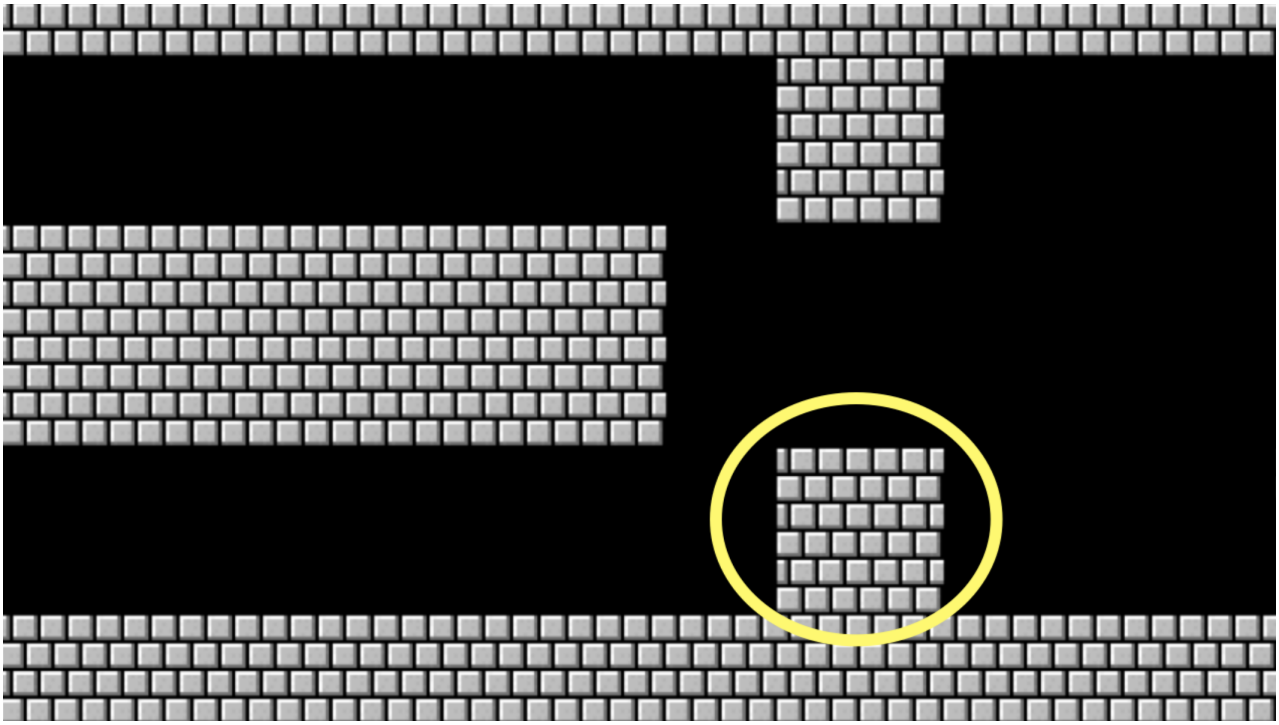
Notice that you can override the behavior of the `print` function to stay on the same line as the previous print.

- Similar in spirit to previous iterations, we can further simplify this program:

```
# Prints a row of 4 question marks without a loop

print("?" * 4)
```

Notice that we can utilize `*` to multiply the print statement to repeat `4` times.

- What about a large block of bricks?



- To implement the above, you can modify your code as follows:

```
# Prints a 3-by-3 grid of bricks with loops

for i in range(3):
    for j in range(3):
        print("#", end="")
    print()
```

Notice how one `for` loop exists inside another. The `print` statement adds a new line at the end of each row of bricks.

- You can learn more about the `print` function in the Python documentation (https://docs.python.org/3/library/functions.html#print)

Scores

- `list`s are a data structure within Python.
- `list`s have built in methods or functions within them.
- For example, consider the following code:

```
# Averages three numbers using a list and a loop

from cs50 import get_int

# Get scores
scores = []
for i in range(3):
    score = get_int("Score: ")
    scores.append(score)
```

```
# Print average
average = sum(scores) / len(scores)
print(f"Average: {average}")
```

Notice that you can use the built-in `append` method, whereby you can `append` the score to the list. Also notice that we use the `sum` function to add all elements in the list.

- You can even ultilize the following syntax:

```
# Averages three numbers using a list and a loop with + operator

from cs50 import get_int

# Get scores
scores = []
for i in range(3):
    score = get_int("Score: ")
    scores += [score]

# Print average
average = sum(scores) / len(scores)
print(f"Average: {average}")
```

Notice that `+=` is utilized to append the score to the list. In this case we place square brackets around `score` because only a `list` can be added to another `list` using `+` or `+=`.

- You can learn more about lists in the Python documentation (https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range)
- You can also learn more about `len` in the Python documentation (https://docs.python.org/3/library/functions.html#len)

Uppercase

- Similarly, consider the following code:

```
# Uppercases string one character at a time

before = input("Before: ")
print("After:  ", end="")
for c in before:
    print(c.upper(), end="")
print()
```

Notice that each character is uppercased one at a time.

- Python has a built-in method for `str`s. You could modify your code as follows:

```
# Uppercases string all at once

before = input("Before: ")
```

```
after = before.upper()
print(f"After:  {after}")
```

Notice the `upper` method is utilized to uppercase the entire string at once.

Greet

- As with C, you can also utilize command-line arguments. Consider the following code:

```
# Prints a command-line argument

from sys import argv

if len(argv) == 2:
    print(f"hello, {argv[1]}")
else:
    print("hello, world")
```

Notice that `argv[1]` is printed using a *formatted string*, noted by the `f` present in the `print` statement.

- You can print all the arguments in `argv` as follows:

```
# Printing command-line arguments, indexing into argv

from sys import argv

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

Notice that the above will not present the word `python` if executed, and the first argument will be the name of the file you are running. You can think of the word `python` as being analogous to `./` when we were running programs in C.

- You can slice pieces of lists away. Consider the following code:

```
# Printing command-line arguments using a slice

from sys import argv

for arg in argv[1:]:
    print(arg)
```

Notice that executing this code will result in the name of the file you are running being sliced away.

- You can learn more about the `sys` library in the Python documentation (https://docs.python.org/3/library/sys.html)

Exit Status

- The `sys` library also has built-in methods. We can use `sys.exit(i)` to exit the program with a specific exit code:

```
# Exits with explicit value, importing sys

import sys

if len(sys.argv) != 2:
    print("Missing command-line argument")
    sys.exit(1)

print(f"hello, {sys.argv[1]}")
sys.exit(0)
```

  Notice that dot-notation is used to utilize the built-in functions of `sys`.

Search

- Python can also be utilized to search. In your terminal window, type `code names.py` and write code as follows:

```
# Implements linear search for names

import sys

# A list of names
names = ["Bill", "Charlie", "Fred", "George", "Ginny", "Percy", "Ron"]

# Ask for name
name = input("Name: ")

# Search for name
for n in names:
    if n == name:
        print("Found")
        sys.exit(0)

print("Not found")
sys.exit(1)
```

  Notice that this code functions. Indeed, it implements a linear search.

- You can utilize the built-in abilities of Python as follows:

```
# Implements linear search for names using `in`

import sys
```

```
# A list of names
names = ["Bill", "Charlie", "Fred", "George", "Ginny", "Percy", "Ron"]

# Ask for name
name = input("Name: ")

# Search for name
if name in names:
    print("Found")
    sys.exit(0)

print("Not found")
sys.exit(1)
```

Notice that the `in` preposition is utilized. Python understands how to implement the lower-level code to do a linear search.

Phonebook

- Recall that a *dictionary* or `dict` is a collection of *key* and *value* pairs.
- You can implement a dictionary in Python as follows:

```
# Implements a phone book

from cs50 import get_string

people = {
    "Carter": "+1-617-495-1000",
    "David": "+1-949-468-2750"
}

# Search for name
name = get_string("Name: ")
if name in people:
    print(f"Number: {people[name]}")
```

Notice that the dictionary is implemented using curly braces. Then, the statement `if name in people` searches to see if the `name` is in the `people` dictionary. Further, notice how, in the `print` statement, we can index into the people dictionary using the value of `name`. Very useful!

- Python has done their best to get to *constant time* using their built-in searches.

Comparison

- We can implement comparisons as follows in Python:

```
# Compares two strings

# Get two strings
s = input("s: ")
t = input("t: ")

# Compare strings
if s == t:
    print("Same")
else:
    print("Different")
```

Notice how Python utilizes the `==` to be able to compare two variables. Further, notice that Python allows you to compare two strings without examining strings character by character using pointers as in C.

Swap

- Further, we can implement a program that swaps values as we did in C. Consider the following code in Python:

```
# Swaps two integers

x = 1
y = 2

print(f"x is {x}, y is {y}")
x, y = y, x
print(f"x is {x}, y is {y}")
```

Notice that each value is swapped, using some very *Pythonic* syntax `x, y = y, x`.

CSV

- You can also utilize Python to engage with CSV files. Consider the following program called `phonebook.py`:

```
# Saves names and numbers to a CSV file

import csv

# Get name and number
name = input("Name: ")
number = input("Number: ")

# Open CSV file
with open("phonebook.csv", "a") as file:
```

```
    # Print to file
    writer = csv.writer(file)
    writer.writerow([name, number])
```

Notice that utilizing the `with` block of code, with the `writer` and its work happening below it indented, prevents us from needing to `close` our file once finished.

- Commonly, CSV files have columns that carry specific names. A `DictWriter` can used to create the CSV file and assign specific names to each column. Consider the following modification to our code:

```
# Saves names and numbers to a CSV file using a DictWriter

import csv

# Get name and number
name = input("Name: ")
number = input("Number: ")

# Open CSV file
with open("phonebook.csv", "a") as file:

    # Print to file
    writer = csv.DictWriter(file, fieldnames=["name", "number"])
    writer.writerow({"name": name, "number": number})
```

Notice the `name` and `number` columns are defined in the penultimate row of code, and values are added in the final line.

- You can learn more about the CSV files in Python in the Python documentation (https://docs.python.org/3/library/csv.html)

Speech

- Using a third-party library, Python can do text-to-speech.

```
# Says hello to someone

import pyttsx3

engine = pyttsx3.init()
name = input("What's your name? ")
engine.say(f"hello, {name}")
engine.runAndWait()
```

- Further, you can run the following code:

```
# Says "This was CS50"

import pyttsx3
```

```
engine = pyttsx3.init()
engine.say("This was CS50")
engine.runAndWait()
```

Summing Up

In this lesson, you learned how the building blocks of programming from prior lessons can be implemented within Python. Further, you learned about how Python allowed for more simplified code. Also, you learned how to utilize various Python libraries. In the end, you learned that your skills as a programmer are not limited to a single programming language. Already, you are seeing how you are discovering a new way of learning through this course that could serve you in any programming language – and, perhaps, in nearly any avenue of learning! Specifically, we discussed…

- Python
- Variables
- Conditionals
- Loops
- Types
- Libraries
- Dictionaries
- Command-line arguments
- Regular expressions

See you next time!