This is CS50x

OpenCourseWare

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/)
(https://www.linkedin.com/in/malan/) (https://www.quora.com/profile/David-J-Malan) (https://twitter.com/davidjmalan)

Lecture 6

Python Basics

- Today we'll learn a new programming language called Python, and remember that one of the overall goals of the course is not learning any particular languages, but how to program in general.
- Source code in Python looks a lot simpler than C, but is capable of solving problems in fields like data science. In fact, to print "hello, world", all we need to write is:

```
print("hello, world")
```

  - Notice that, unlike in C, we don't need to import a standard library, declare a `main` function, specify a newline in the `print` function, or use semicolons.
- Python is an interpreted language, which means that we actually run another program (an interpreter) that reads our source code and runs it top to bottom. For example, we can save the above as `hello.py`, and run the command `python hello.py` to run our code, without having to compile it.
- We can get strings from a user:

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

  - We create a variable called `answer`, without specifying the type (the interpreter determins that from context for us), and we can easily combine two strings with the `+` operator before we pass it into `print`.
  - We can also pass in multiple arguments to `print`, with `print("hello,", answer)`, and it will automatically join them with spaces for us too.
  - `print` also accepts format strings like `f"hello, {answer}"`, which substitutes variables inside curly braces into a string.
- We can create variables with just `counter = 0`. To increment a variable, we can use `counter = counter + 1` or `counter += 1`.
- Conditions look like:

```
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")
```

  - Unlike in C and JavaScript (whereby braces `{ }` are used to indicate blocks of code), the exact indentation of each line is what determines the level of nesting in Python.
  - And instead of `else if`, we just say `elif`.
- Boolean expressions are slightly different, too:

```
while True:
    print("hello, world")
```

- We can write a loop with a variable:

```
i = 3
while i > 0:
    print("cough")
    i -= 1
```

- We can also use a `for` loop, where we can do something for each element in a list:

```
for i in [0, 1, 2]:
    print("cough")
```

  - Lists in Python are like arrays in C, but they can grow and shrink easily with the interpreter managing the implementation and memory for us.
  - This `for` loop will set the variable `i` to the first element, `0`, run, then to the second element, `1`, run, and so on.
  - And we can use a special function, `range`, to get some number of values, as in `for i in range(3)`. This will give us `0`, `1`, and `2`, for a total of thee values.
- In Python, there are many data types:
  - `bool`, `True` or `False`
  - `float`, real numbers
  - `int`, integers
  - `str`, strings
  - `range`, sequence of numbers
  - `list`, sequence of mutable values, that we can change or add or remove
  - `tuple`, sequence of immutable values, that we can't change
  - `dict`, collection of key/value pairs, like a hash table
  - `set`, collection of unique values
- docs.python.org (https://docs.python.org) is the official source of documentation, but Google and StackOverflow will also have helpful resources when we need to figure out how to do something in Python. In fact, programmers in the real world rarely know everything in the documentation, but rather how to find what they need when they need it.

Examples

- We can blur an image with:

```
from PIL import Image, ImageFilter

before = Image.open("bridge.bmp")
after = before.filter(ImageFilter.BLUR)
after.save("out.bmp")
```

  - In Python, we include other libraries with `import`, and here we'll `import` the `Image` and `ImageFilter` names from the `PIL` library.
  - It turns out, if we look for documention for the `PIL` library, we can use the next three lines of code to open an image called `bridge.bmp`, run a blur filter on it, and save it to a file called `out.bmp`.
  - And we can run this with `python blur.py` after saving to a file called `blur.py`.
- We can implement a dictionary with:

```
words = set()

def check(word):
    if word.lower() in words:
        return True
    else:
        return False

def load(dictionary):
    file = open(dictionary, "r")
    for line in file:
        words.add(line.rstrip("\n"))
    file.close()
    return True

def size():
    return len(words)

def unload():
    return True
```

  - First, we create a new set called `words`. Then, for `check`, we can just ask ` `if word.lower() in words`. For `load`, we open the file and use `words.add` to add each line to our set. For `size`, we can use `len` to count the number of elements in our set, and finally, for `unload`, we don't have to do anything!

- It turns out, even though implementing a program in Python is simpler for us, the running time of our program in Python is slower than our program in C since our interpreter has to do more work for us. So, depending on our goals, we'll also have to consider the tradeoff of human time of writing a program that's more efficient, versus the running time of the program.
- In Python, we can too include the CS50 library, but our syntax will be:

```
from cs50 import get_string
```

  - Notice that we specify the functions we want to use.

- Now we can get strings from a user:

```
from cs50 import get_string

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

- We can substitute expressions into our format strings, too:

```
from cs50 import get_int

age = get_int("What's your age?\n")
print(f"You are at least {age * 365} days old.")
```

- And we can demonstrate conditions:

```
from cs50 import get_int

x = get_int("x: ")
y = get_int("y: ")

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")
```

- To check conditions, we can say:

```
from cs50 import get_string

s = get_string("Do you agree?\n")

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

  - Python doesn't have chars, so we can check them as strings directly.

- We can also say `if s in ["Y", "y"]:` , or `if s.lower() in ["y"]:` . It turns out that strings in Python are like structs in C, where we have not only variables but functions that we can call. For example, given a string `s` , we can call its `lower` function with `s.lower()` to get the lowercase version of the string.

- We can improve versions of `cough` , too:

```
print("cough")
print("cough")
print("cough")
```

  - We don't need to declare a `main` function, so we just write the same line of code three times.

- But we can do better:

```
for i in range(3):
    cough()

def cough():
    print("cough")
```

  - Notice that we don't need to specify the return type of a new function, which we can define with `def` .
  - But this causes an error when we try to run it: `NameError: name 'cough' is not defined` . It turns out that we need to define our function before we use it, so we can either move our definition of `cough` to the top, or create a main function:

```
def main():
    for i in range(3):
        cough()

def cough():
    print("cough")

main()
```

  - Now, by the time we actually call our `main` function, the `cough` function will have been read by our interpreter.

- Our functions can take inputs, too:

```
def main():
    cough(3)

def cough(n):
    for i in range(n):
        print("cough")

main()
```

- We can define a function to get a positive integer:

```
from cs50 import get_int

def main():
    i = get_positive_int()
    print(i)

def get_positive_int():
    while True:
        n = get_int("Positive Integer: ")
        if n > 0:
            break
    return n

main()
```

  - Since there is no do-while loop in Python as there is in C, we have a `while` loop that will go on infinitely, but we use `break` to end the loop as soon as `n > 0` . Then, our function will just `return n` .
  - Notice that variables in Python have function scope by default, meaning that `n` can be initialized within a loop, but still be accessible later in the function.

- We can print out a row of question marks on the screen:

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

  - When we print each block, we don't want the automatic new line, so we can pass a parameter, or named argument, to the `print` function. Here, we say `end=""` to specify that nothing should be printed at the end of our string. Then, after we print our row, we can

call `print` to get a new line.

- We can also "multiply" a string and print that directly with: `print("?" * 4)`.
- We can print a column with a loop:

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

- And without a loop: `print("#\n" * 3, end="")`.
- We can implement nested loops:

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

- We don't need to use the `get_string` function from the CS50 library, since we can use the `input` function built into Python to get a string from the user. But if we want another type of data, like an integer, from the user, we'll need to cast it with `int()`.
- But our program will crash if the string isn't convertable to an integer, so we can use `get_string` which will just ask again.
- In Python, trying to get an integer overflow actually won't work:

```
from time import sleep

i = 1
while True:
    print(i)
    sleep(1)
    i *= 2
```

  - We call the `sleep` function to pause our program for a second between each iteration.
  - This will continue until the integer can no longer fit in your computer's memory.
- Floating-point imprecision, too, can be prevented by libraries that can represent decimal numbers with as many bits as are needed.
- We can make a list:

```
scores = []
scores.append(72)
scores.append(73)
scores.append(33)

print(f"Average: {sum(scores) / len(scores)}")
```

  - With `append`, we can add items to our list, using it like a linked list.
  - We can also declare a list with some values like `scores = [72, 73, 33]`.
- We can iterate over each character in a string:

```
from cs50 import get_string

s = get_string("Input:  ")
print("Output: ", end="")
for c in s:
    print(c, end="")
print()
```

  - Python will get each character in the string for us.
- To make a string uppercase, too, we can just call `s.upper()` to get the uppercase version of the entire string, without having to iterate over each character ourselves.

More features

- We can take command-line arguments with:

```
from sys import argv

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

  - Since `argv` is a list of strings, we can use `len()` to get its length, and `range()` for a range of values that we can use as an index for each element in the list.
- But we can also let Python iterate over the list for us:

```
from sys import argv

for arg in argv:
    print(arg)
```

- We can return exit codes when our program exits, too:

```
from sys import argv, exit

if len(argv) != 2:
    print("missing command-line argument")
    exit(1)
print(f"hello, {argv[1]}")
exit(0)
```

  - We import the `exit` function, and call it with the code we want our program to exit with.

- We can implement linear search by just checking each element in a list:

```
import sys

names = ["EMMA", "RODRIGO", "BRIAN", "DAVID"]

if "EMMA" in names:
    print("Found")
    sys.exit(0)
print("Not found")
sys.exit(1)
```

- If we have a dictionary, a set of key:value pairs, we can also check each key:

```
import sys

people = {
    "EMMA": "617-555-0100",
    "RODRIGO": "617-555-0101",
    "BRIAN": "617-555-0102",
    "DAVID": "617-555-0103"
}

if "EMMA" in people:
    print(f"Found {people['EMMA']}")
    sys.exit(0)
print("Not found")
sys.exit(1)
```

  - Notice that we can get the value of of a particular key in a dictionary with `people['EMMA']`. Here, we use single quotes (both single and double quotes are allowed, as long they match for a string) to differentiate the inner string from the outer string.
  - And we declare dictionaries with curly braces, `{}`, and lists with brackets `[]`.

- In Python, we can compare strings directly with just `==`:

```
from cs50 import get_string

s = get_string("s: ")
t = get_string("t: ")

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

- Copying strings, too, works without any extra work from us:

```
from cs50 import get_string

s = get_string("s: ")

t = s

t = t.capitalize()

print(f"s: {s}")
print(f"t: {t}")
```

- Swapping two variables can also be done by assigning both values at the same time:

```
x = 1
y = 2

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

Files

- Let's open a CSV file:

```
import csv
from cs50 import get_string

file = open("phonebook.csv", "a")

name = get_string("Name: ")
number = get_string("Number: ")

writer = csv.writer(file)
writer.writerow((name, number))

file.close()
```

  - It turns out that Python also has a `csv` package (library) that helps us work with CSV files, so after we open the file for appending, we can call `csv.writer` to create a `writer` from the file and then `writer.writerow` to write a row. With the inner parentheses, we're creating a tuple with the values we want to write, so we're actually passing in a single argument that has all the values for our row.
- We can use the `with` keyword, which will helpfully close the file for us:

```
...
with open("phonebook.csv", "a") as file:
    writer = csv.writer(file)
    writer.writerow((name, number))
```

New features

- A feature of Python that C does not have is **regular expressions**, or patterns against which we can match strings. For example, its syntax includes:
  - `.`, for any character
  - `.*`, for 0 or more characters
  - `.+`, for 1 or more characters
  - `?`, for something optional
  - `^`, for start of input
  - `$`, for end of input
- For example, we can match strings with:

```
import re
from cs50 import get_string

s = get_string("Do you agree?\n")

if re.search("^y(es)?$", s, re.IGNORECASE):
    print("Agreed.")
elif re.search("^no?$", s, re.IGNORECASE):
    print("Not agreed.")
```

  - First, we need the `re` package, or library, for regular expressions.
  - Then, for `y` or `yes`, we have the regular expression `^y(es)?$`. We want to make sure that the string starts with `y`, and optionally has `es` immediately after the `y`, and then ends.
  - Similarly, for `n` and `no`, we want our string to start, have the letter `n`, and optionally the letter `o` next, and then end. The regular expression for that would be `^no?$`.
  - We pass in another argument, `re.IGNORECASE`, to ignore the casing of the letters in the string.
  - If neither regular expression matches, we wouldn't print anything.
- On our own Mac or PC, we can open a terminal after installing Python, and use the microphone to convert our speech to text:

```
import speech_recognition

recognizer = speech_recognition.Recognizer()
with speech_recognition.Microphone() as source:
    print("Say something!")
    audio = recognizer.listen(source)

print("Google Speech Recognition thinks you said:")
print(recognizer.recognize_google(audio))
```

  - It turns out that there's another library we can download, called `speech_recognition`, that can listen to audio and convert it to a string.
- And now, we can match on the audio to print something else:

```
...
words = recognizer.recognize_google(audio)

# Respond to speech
if "hello" in words:
    print("Hello to you too!")
elif "how are you" in words:
    print("I am well, thanks!")
elif "goodbye" in words:
    print("Goodbye to you too!")
else:
    print("Huh?")
```

- We can even use regular expressions, to match on part of a string:

```
...
words = recognizer.recognize_google(audio)

matches = re.search("my name is (.*)", words)
if matches:
    print(f"Hey, {matches[1]}.")
else:
    print("Hey, you.")
```

  - Here, we can get all the characters after `my name is` with `.*`, and print it out.
- We run detect.py and faces.py (https://cdn.cs50.net/2019/fall/lectures/6/src6/6/faces/), which finds each face (or even a specific face) in a photo.
- qr.py (https://cdn.cs50.net/2019/fall/lectures/6/src6/6/qr/) will also generate a QR code to a particular URL.