This is CS50x

OpenCourseWare

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.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

Last Time

- We learned some basics about the internet, and technologies like:
  - TCP/IP, protocols by which computers can send each other messages across a network of many computers, using IP adc and port numbers.
  - HTTP, a protocol by which browsers, and other programs, can make a request for a webpage (or other content) from a s
  - URLs, including a domain name and parameters like `?q=cats`, to pass along additional inputs to a server.
  - HTTP status codes, like 404 Not Found, which shows us an error page, and 301 Moved Permanently, which redirects us right URL if a website has moved.
  - HTML and CSS, languages by which we can format and stylize webpages.
  - JavaScript and the DOM, Document Object Model, by which we can change nodes in a tree representation of an HTML thereby changing the page itself.

Python

- Python is another programming language, but it is interpreted (run top to bottom by an interpreter, like JavaScript) and high (including features and libraries that are more powerful).
- For example, we can implement the entire `resize` program in just a few lines with Python:

```
import sys
from PIL import Image

if len(sys.argv) != 4:
    sys.exit("Usage: python resize.py n infile outfile")

n = int(sys.argv[1])
infile = sys.argv[2]
outfile = sys.argv[3]

inimage = Image.open(infile)
width, height = inimage.size
outimage = inimage.resize((width * n, height * n))

outimage.save(outfile)
```

- First, we `import` (like `include`) a `sys` library (for command-line arguments) and an `Image` library.
- We check that there are the right number of command-line arguments with `len(sys.argv)`, and then create some v `n`, `infile`, and `outfile`, without having to specify their types.
- Then, we use the Image library to open the input image, getting its width and height, resizing it with a `resize` functi finally saving it to an output file.
- Let's take a look at some new syntax. In Python, we can create variables with just `counter = 0`. To increment a variable, v use `counter = counter + 1` or `counter += 1`.
- Conditions look like:

```
if x < y:
    something
elif:
    something
else:
    something
```

  - Unlike in C and JavaScript (whereby braces `{` `}` are used for blocks of code), the exact indentation of each line is wh determines the level of nesting in Python.
- Boolean expressions are slightly different, too:

```
while True:
    something
```

- Loops can be created with another function, `range`, that, in the example below, returns a range of numbers from 0, up to b including 50:

```
for i in range(50):
    something
```

- In Python, we'll start by looking at just a few data types:
  - `bool`, `True` or `False`
  - `float`, real numbers
  - `int`, integers
  - `str`, strings
  - `dict`, a dictionary of key-value pairs, that act like hash tables
  - `list`, like arrays, but can automatically resize
  - `range`, range of values
  - `set`, a collection of unique things
  - `tuple`, a group of two or more things
- In Python, we can too include the CS50 library, but our syntax will be:

```
from cs50 import get_float, get_int, get_string
```

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

- In Python, we can run our program without compiling it with `python hello.py` (or whatever the name of our file is).
  - `python` is name of the program that we're actually running at the command line, and it is an interpreter which can re source code (written in the language Python) and run it, one line at a time. (Technically, there is a compiler that turns c source code into something called bytecode that the interpreter actually runs, but that is abstracted away for us.)

Data types in Python

- Our first `hello.py` program is just:

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

  - Notice that we didn't need a `main` function, or anything that we needed to import for the `print` function. The `prin` function in Python also adds a new line for us automatically.
  - Now we can run it with `python hello.py`.

- We can get strings from a user:

```
from cs50 import get_string

s = get_string("Name: ")
print("hello,", s)
```

  - We create a variable called `s`, without specifying the type, and we can pass in multiple variables into the `print` fun which will print them for us on the same line, separated by a space automatically.
  - To avoid the extra spaces, we can put variables inside a string similar to how they are included in C: `print(f"hello,` Here, we're saying that the string `hello, {s}` is a formatted string, with the `f` in front of the string, and so the varia will be substituted in the string. And we don't need to worry about the variable type; we can just include them inside s

- We can do some math, too:

```
from cs50 import get_int

x = get_int("x: ")

y = get_int("y: ")

print(f"x + y = {x + y}")
print(f"x - y = {x - y}")
print(f"x * y = {x * y}")
print(f"x / y = {x / y}")
print(f"x mod y = {x % y}")
```

  - Notice that expressions like `{x + y}` will be evaluated, or calculated, before it's substituted into the string to be prin
  - By running this program, we see that everything works as we might expect, even dividing two integers to get a floating value. (To keep the old behavior of always returning a truncated integer with division, there is the `//` operator.)

- We can experiment with floating-point values:

```
from cs50 import get_float

x = get_float("x: ")

y = get_float("y: ")

z = x / y

print(f"x / y = {z}")
```

- We see the following when we run this program:

```
$ python floats.py
x: 1
y: 10
x / y = 0.1
```

- We can print more decimal places with syntax like `print(f"x / y = {z:.50f}")`:

```
x / y = 0.10000000000000005551115123125782702118158340454101
```

  - It turns out that Python still has floating-point imprecision by default, but there are some libraries that will use m memory to store decimal values more precisely.

- We can see if Python has integer overflow:

```
from time import sleep

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

  - We use the `sleep` function to pause our program for one second, but double `i` over and over. And it turns out that i in Python can be as big as memory allows, so we won't experience overflow for a much longer time.

Programming in Python

- Let's take a closer look at conditions:

```
from cs50 import get_int

# Get x from user
x = get_int("x: ")

# Get y from user
y = get_int("y: ")

# Compare x and 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")
```

  - Notice that we use consistent indentation, but we don't need parentheses or braces for our conditions.
  - Comments, too, start with just a single `#` character.

- We can compare strings the way we might expect:

```
from cs50 import get_char

# Prompt user for answer
c = get_string("Answer: ")

# Check answer
if c == "Y" or c == "y":
    print("yes")
elif c == "N" or c == "n":
    print("no")
```

- Strings can be compared directly, and Boolean expressions can include the words `and` and `or`.

- We can write functions in Pythons like this:

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


def cough():
    """Cough once"""
    print("cough")


if __name__ == "__main__":
    main()
```

  - We use the `def` keyword to define a function `cough`, indicating that it takes no parameters, or inputs, by using just `
    call it from our `main` function. Notice that all the code for each function is indented additionally, instead of surrounde
    braces.
  - Then, at the below, we use a special line `if __name__ == "__main__":` to call our `main` function when our progra
    run. This way, the interpreter will know about the `cough` function by the time `main` actually calls it. We could also c
    `cough` directly, instead of `main`, though that would be unconventional in Python. (Instead, we want to try to be "Pyth
    following the styles and patterns encouraged by the language and its community.)

- We can add parameters and loops to our `cough` function, too:

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


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


if __name__ == "__main__":
    main()
```

  - `n` is a variable that can be passed into `cough`, which we can also pass into `range`. And notice that we don't specify
    Python, so `n` can be of any data type (and can even be assigned to have a value of another type). It's up to us, the prog
    to use this great power with great responsibility.

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

```
from cs50 import get_int


def main():
    i = get_positive_int("Positive integer: ")
    print(i)


def get_positive_int(prompt):
    while True:
        n = get_int(prompt)
        if n > 0:
            break
    return n


if __name__ == "__main__":
    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 to end the loop if `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 st accessible later in the function.

- We can print each character in a string and capitalize them:

```
from cs50 import get_string

s = get_string()
for c in s:
    print(c.upper(), end="")
print()
```

  - Notice that we can easily iterate over characters in a string with something like `for c in s`, and we print the upper version of each character with `c.upper()`. Strings in Python are objects, like a data structure with both the value it s well as built-in functions like `.upper()` that we can call.
  - Finally, we pass in another argument to the `print` function, `end=""`, to prevent a new line from being printed each Python has named arguments, where we can name arguments that we can pass in, in addition to positional arguments on the position they are in the list. With named arguments, we can pass in arguments in different orders, and omit opti arguments entirely. Notice that this example is labeled with `end`, indicating the string that we want to end each print with. By passing in an empty string, `""`, nothing will be printed after each character. Before, when we called `print` the `end` argument, the function used `\n` as the default for `end`, which is how we got new lines automatically.
- We can get the length of the string with the `len()` function.

```
from cs50 import get_string

s = get_string("Name: ")
print(len(s))
```

- We'll be using version 3 of Python, which the world is starting to use more and more, so when searching for documentation want to be sure that it's for the right version.
- We can take command-line arguments with:

```
from sys import argv

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

- We check the number of arguments by looking at the length of `argv`, a list of arguments, and if there is 2, we print th second one. Like in C, the first command-line argument is the name of the program we wrote, rather than the word `py` which is technically the name of the program we run at the command-line.
- We can print each argument in the list:

```
from sys import argv

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

  - This will iterate over each element in the list `argv`, allowing us to use it as `s`.
- And we can iterate over each character, of each argument:

```
from sys import argv

for s in argv:
    for c in s:
        print(c)
    print()
```

- We can swap two variables in Python just by reversing their orders:

```
x = 1
y = 2

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

  - Here, we're using `x, y = y, x` to set `x` to `y` at the same time as setting `y` to `x`.

- We can create a list and add to it:

```
from cs50 import get_int

numbers = []

# Prompt for numbers (until EOF)
while True:

    # Prompt for number
    number = get_int("number: ")

    # Check for EOF
    if not number:
        break

    # Check whether number is already in list
    if number not in numbers:

        # Add number to list
        numbers.append(number)

# Print numbers
print()
for number in numbers:
    print(number)
```

  - Here, we create a empty list called `numbers` with `numbers = []`, and we get a `number` from the user. If that `numb` not already in our list, we add it to our list. We can use `not in` to check if a value is (not) in a list, and `append` to ad value to the end of a list.

- We can create our own data structures, objects:

```
from cs50 import get_string

# Space for students
students = []

# Prompt for students' names and dorms
for i in range(3):
    name = get_string("name: ")
    dorm = get_string("dorm: ")
    students.append({"name": name, "dorm": dorm})

# Print students' names and dorms
for student in students:
    print(f"{student['name']} is in {student['dorm']}.")
```

- We create a list called `students`, and after we get some input from the user, we append a dictionary of key-value pai `{"name": name, "dorm": dorm}`, to that list. Here, `"name"` and `"dorm"` are the keys, and we want their values to variables we gathered as input. Then, we can later access each object's values with `student['name']` or `student['` to print them out. In Python, we can index into dictionaries with words or strings, as opposed to just numeric indexes i

- Let's print four question marks, one at a time:

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

- We can print a vertical bar of hash marks, too:

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

- And we can print a square with a nested loop:

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

- Now we can revisit `resize.py`, and it might make more sense to us now:

```
from PIL import Image
from sys import argv

if len(sys.argv) != 4:
    sys.exit("Usage: python resize.py n infile outfile")

n = int(sys.argv[1])
infile = sys.argv[2]
outfile = sys.argv[3]

inimage = Image.open(infile)
width, height = inimage.size
outimage = inimage.resize((width * n, height * n))

outimage.save(outfile)
```

- We import the Image library from something called PIL, a free open-source library that we can download and install (v doesn't come with Python by default).
- Then, we import `argv` from the system library, and we check our arguments, storing them as `n`, `infile`, and `outf` converting the string input for `n` into an `int` as we do so.
- By reading the documentation for Python and the Image library, we can open files as an image, getting its `size` and `resize` function on it to get another image, which we can then `save` to another file.

- Let's look at another example, a spell-checker in Python:

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

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

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

  - The functions for `dictionary.py` are pretty straightforward, since all we need is a `set()`, a collection into which w load unique values. In `load`, we open the `dictionary` file, and add each line in the file as a word (without the newl character).
  - For `check`, we can just return whether `word` is in `words`, and for `size`, we can just return the length of `words`. F we don't need to do anything to `unload`, since Python manages memory for us.
- By having used C first, we have an understanding (and appreciation!) for the abstractions that a higher-level language like provides us. Indeed, if we run some tests for performance, a speller implementation in Python might be 1.5x slower, and so depending on the application, this may or may not be important enough to justify the human time it might take to write a in a lower-level language like C, which might run much faster or require less memory.