Week 14 - Monday

SLD 2600

Python

Python

- Python is a programming language invented by Dutch programmer Guido Van Rossum
- Many language design decisions are because "Guido likes it that way"
- He worked for Google for a while and is retired now
- Python is free!
  - You can download it for Windows, Mac, or Linux at https://www.python.org/
- Python has become popular in lots of fields because it has simple, English-like syntax



Image from Wikipedia, available: http://en.wikipedia.org/wiki/File:Guido_van_Rossum_OSCON_2006.jpg
Creative Commons Attribution-Share Alike 2.0 Generic License

Hello, world!

- The first program in many programming languages is called Hello, world
- It's a test to see if text input and output are working
- In Python, you can type the following to write the Hello, world program

```
print("Hello, world!")
```

- It will print **`Hello, world!`** on the next line

Variables

Variables

- A variable is a common abstraction in computer programs
- Think of a variable as a named box that holds a value
- Depending on the situation (and the rules of the language) a variable could hold:
  - An integer like 14
  - A fractional number like 3.14
  - Text like "Where late the sweet birds sang"
  - Lists of the above
  - More complicated combinations of the above data
- Unlike in math, we can change the value in the variable as the program progresses

Variables in Python

- In Python, we assign a value to a name and that's it

```
person = "Walter White"
```

- The = sign is the assignment operator
- Think of it as an arrow that points left, storing whatever is on the right into the variable
- We can use whatever is stored in a variable like it's a value

```
print(person)
```

Repetition

Loops in Python

- **Good news:** There are only two loops to learn in Python
- The **`while`** loop runs as long as a condition is true

```
countdown = 10
while countdown > 0:
      print(countdown)
      countdown = countdown - 1
print("Blast-off!")
```

- In this case, when **`countdown`** becomes **0**, the loop stops (since **0** is not greater than **0**)

For loop

- Perhaps the most common loop in Python is the **`for`** loop, which will loop over everything in a list of things

```
for letter in "Help me!":
    print(letter)
```

- This code prints each letter in the text on a separate line

Repeating a set number of times

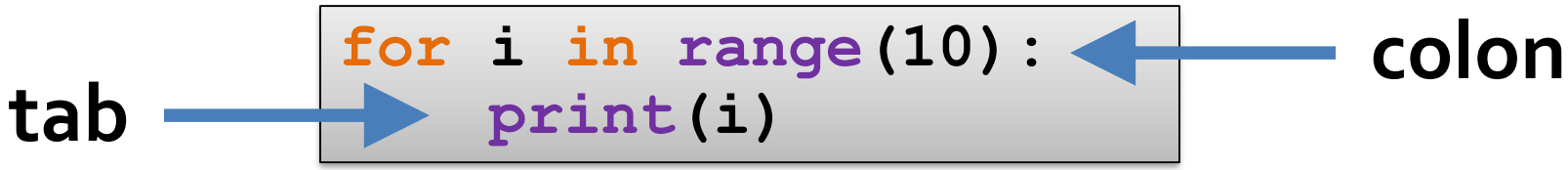
- If you want to do something, say, 10 times in Python, you can use a **`for`** loop
- But you need to call the **`range()`** method to generate a list of numbers for you

```
for i in range(10):
    print(i)
```

- In this case, the loop will run 10 times, and the numbers **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, and **9** will print out

Annoying syntax

- The first line of every loop ends with a colon (:)
- If you don't put it there, Python will not run your code
- The stuff inside of a loop needs to be indented with a tab
  - That's how you know what should be repeated and what is after the loop



- If you type the colon, most Python editors automatically indent the next line for you

Choices in Python

Making a choice

- Sometimes you want to make a choice based on a condition
- If the condition is true, you do some task
- If it's false, you don't do anything
- Example:
  - If it's raining:
    - I'll bring an umbrella

If statements

- In Python, the primary way to make a choice is with an **`if`** statement
- An if statement looks like a while loop, except that it uses the keyword **`if`** instead of **`while`**

```
if condition:
    do things
```

- Whatever code is under the **`if`** and tabbed in will only be executed if the condition is true
- Don't forget the colon after the condition!

Example 1

- We can write a program that can check to see if someone knows a password

```
if phrase == "open sesame":
    print("Welcome, Ali Baba!")
```

- If the **"open sesame"** is stored into **phrase**, the message will be printed
- If anything else is typed in, nothing will print

Comparison

- You might have noticed that we used `==` to compare the input to the text `"open sesame"`
- We cannot use `=` because that operator is used to assign a value
- We will usually use one of the operators on the right for comparison

Operator	Meaning
==	Equals
!=	Does not equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Example 2

- We will often use one of the inequality operators to cover a range of values
- For example, we can see if a bank balance is negative

```
if balance < 0:
    print("Your account is overdrawn!")
```

Else

- It is often the case that we want to do one thing if a certain condition is met and something else if it is not
- Example:
  - If it's raining:
    - I'll bring an umbrella
  - Otherwise:
    - I'll eat lunch in the park
- Python uses the keyword **`else`** to mark code that will be executed if the condition was false

Example 3

- Using **`else`**, we can give a different message if the person doesn't know the password

```
if phrase == "open sesame":
      print("Welcome, Ali Baba!")
else:
      print("Access denied!")
```

- If anything other than **`"open sesame"`** is stored in **`phrase`**, the message **`Access denied!`** will be printed

Example 4

- Using **`else`**, we can also give a message if the account is not overdrawn

```
if balance < 0:
      print("Your account is overdrawn!")
else:
      print("You have $" + str(balance))
```

Things to watch out for

- The condition of an **`if`** must be followed by a colon (**`:`**)
- Never use **`=`** in a condition (always use **`==`**)
- It's easy to be off by one (**`x < 5`** is not the same as **`x <= 5`**)
- An **`else`** also has a colon after it
- An **`else`** never has a condition
  - It marks what gets done when the **`if`**'s condition is false
- Using an **`else`** is not required
  - But you can't have an **`else`** without an **`if`**

Lists in Python

Lists of data

- Computers are fast, but you need to process a lot of data to get a sense of how fast they are
- Python provides us with strings for text data, which are really a special kind of list
- Python also provides general lists that can hold anything (even other lists)
- Since lists can hold many items, it is common to use loops (especially **`for`** loops) to deal with them

Strings

- Strings hold text
  - They are lists of characters (letters, digits, punctuation, etc.)
- A string literal in Python is text with either single or double quotes around it

```
line = "Stop, collaborate, and listen!"
countdown = 'Launching in 5, 4, 3, 2, 1... Blast off!'
```

- It doesn't matter which you use, but you can't mix them in one string

String operations

- You can use **+** to concatenate two strings together (to get a third string that is both of them stuck together)

```
place = "boon" + "docks"
print(place) #prints boondocks
```

- You can use ***** to get repetitions of a string

```
comment = "yeah " * 3
print(comment) #prints yeah yeah yeah
```

String operations continued

- You can use the **`len()`** function to get the length of a string

```
author = "Thomas Pynchon"
print(len(author)) #prints 14
```

- You can use square brackets to get a particular character in the string
- Indexes start at 0
  - The first character in a string is at 0, the last is at its length - 1

```
movie = "Dr. Strangelove"
print(movie[4]) #prints S
```

Lists

- You can think of a string as a list of characters
- Python provides a way to make lists of other things too
- To make a list, you can put a collection of objects inside square brackets

```
days = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]
```

- They can even be different types

```
stuff = ["Danger!", 3, True, 1.7]
```

Accessing an element

- As with strings, use square brackets and a number to access an element in the list

```
days = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]
bleh = days[0] #contains "Monday"
```

- Like strings, elements are numbered from 0 to the length – 1
- You can use the **`len()`** function to get the length of a list

```
count = len(days)   #contains 7
```

Changing elements in a list

- You can also change the elements in a list using the square bracket notation

```
birds = ["Duck", "Duck", "Duck"]
birds[2] = "Goose"
print(birds) #prints ['Duck', 'Duck', 'Goose']
```

- This is one of the bigger differences between strings and general lists
- You **cannot** change the characters in a string
- You have to make a new string

For loops and lists

- A **for** loop can be used to iterate over all the elements in a list

```
days = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]
for day in days:
    print(day) #prints each day on a line
```

- Since a string is a special kind of list, you can iterate over the characters in them too

```
word = "HEY"
for letter in word:
    print(letter) #H E Y on separate lines
```

Files

Files

- When you turn off your computer, no programs can run on it
- All the data in RAM is gone
- It's useful to be able to store data for long periods of time
- **Files** are a general mechanism for storing data
- Everything you see in a folder on your computer is a file
  - Even the folders themselves are a special kind of file

Opening a file using `with/as`

- Python has syntax that opens a file and closes it when you're done
- The syntax starts with the keyword **`with`**
- Then, code using the file is in an indented block

```
with open('data.txt', 'r') as file:
  # Do the reading you want to do with file
  # Do some calculations
```

- The file is automatically closed after the indented block

File processing

- Files are often read one line at a time
- Python lets us iterate over the file as if it were a list of lines

```
with open('alice.txt', 'r') as story:
  for line in story:
    print(line)
```

Using `split()` with files

- Each line of a file might contain several data fields.
- The **`split()`** method can be used to break a line into a list of fields
- For example, a comma-separated-value (CSV) file divides values with commas

```
with open('data.csv', 'r') as data:
  for line in data:
    for column in line.split(','):
        print(column)
```

Assignment for this class

- Write 250-500 words addressing the following question:
  - Have you ever done any programming before?
  - What kind of programming , if any, is the most common in your field?
  - Do you have any plans to get prepared for the programming you might have to do?

Upcoming

Presentations

- **The next (and final) class meeting is presentations!**
  - Give a presentation on any topic you find interesting
  - Design the presentation to be engaging for the audience of your fellow classmates
  - Create and deliver the presentation using PowerPoint
  - The presentation should not be less than 3 minutes and not more than 4 minutes, including audience questions

Next time…

- Second-year students will meet next week, on November 23, to give presentations
- First-year students will meet in two weeks, on November 30, to give presentations

Reminders

- Assignment 7 is due next Monday
- Don't forget Assignment 5 (for first-year students) or Assignment 4 (for second-year students) in which you have to get a selfie with your mentor