CSCI E-50 WEEK 8

TERESA LEE
[teresa.lee@alumni.utoronto.ca]
October 24, 2017

Agenda

- Python
- Flask

Python: Basics

- A newer language than C. C was first released in 1972, Python in 1991.
- “Interpreted Language”
- Written using the .py file extension and are run via the Python interpreter
- In CS50 ,we use Python 3.

Python: Syntax

- Python doesn't use curly braces or semicolons.
- Scope is determined by indentation (4 spaces to be exact) - STYLE MATTERS!
    - PYTHON STYLE GUIDE
- In Python # is used for comments instead of //.

Resources

- Python’s official documentation (Python 3)
- The Python Language Reference
- The Python Standard Library
- The Python Tutorial

Python: Variables

- Python variables do not have explicit data types
- Python variables do not need to be declared in advance
- Python variables **do** have underlying data types:
  - Number
  - Stirng
  - List
  - Tuple dictionary

C

```
int a = 3;
char b = "c";
int arr[] = {1, 2, 3, 4};
char myString[] = "This is a string";
```

Python

```
a = 3
b = "c"
arr = [1, 2, 3, 4]
myString = "This is a string"
```

Python: Conditionals

- Conditions don't have to be inside parentheses, except for grouping.
- Instead of using curly braces, we use a colon and indentation.
- The conditional is terminated by returning to the previous indentation level.
- &&, || are replaced with and, or. ! is replaced with not.
- else if is shortened to elif

Python: Conditionals

C

```c
if (a != 50)
{
    b = 1;
}
else if (b > 0)
{
    b = 0;
}
else
{
    b = -1;
}
```

Python

```python
if a != 50:
    b = 1
elif b > 0:
    b = 0
else:
    b = -1
```

Python: Loops

- `for` and `while` are the two primary iterating constructs in Python.
- `for`, in particular, has extreme flexibility relative to its C cousin.
- `do-while` does not exist in Python and has to be hacked with a `while True:` and a `break`
- The code subject to a loop is introduced by `:` instead of `{`.
- All code subject to the loop **must** be indented in order things to work as intended.
- The loop is terminated by returning to the previous indentation level.

Python: Loops

c

```c
for (int i = 0; i < 50; i++)
{
    // Do something
}
```

python

```python
for i in range(50):
    # Do something
```

Or:

```python
for i in range(0, 50):
    # Do something
```

Or:

```python
for i in range(0, 50, 1):
    # Do something
```

Python: Loops

C

```c
int i = 0;
while(i < 100)
{
    printf("%i\n", ++i);
}

for(int j = 0; j < 100; j += 2)
{
    printf("%i\n", j);
}
```

Python

```python
i = 0
while i < 100:
    print(i)
    i += 1

for j in range(0, 101, 2):
    print(j)
```

Python: Loops

C

```c
for (int i = 1337; i > 50; i -= 3)
{
    // Do something
}
```

python

```python
for i in range(1337, 50, -3):
    # Do something
```

C

```c
int vals[] = {4, 5, 6, 1, 2, 4, 2, 44, 5};
int sum = 0;
for (int i = 0; i < 9; i++)
{
    sum += vals[i];
}
```

python

```python
vals = [4, 5, 6, 1, 2, 4, 2, 44, 5]
sum = 0
for num in vals:
    sum += num
```

Let’s Look at An Example

`quack.c`

`argv0.c`

`argv1.c`

`argv2.c`

- Lists are the Python version of Arrays.
- They can be dynamically grown or shrunk.
- They can also contain variables of multiple types.

Python: Lists

- To initialize an array you use the square brackets:

  vals = [] - vals is an empty list

  vals = [1, 2, 3] - vals contains the ints 1, 2 and 3.

- To initialize an array containing the number 1 through 100:

  ```
  vals = [for i in range(1, 101)]
  ```

- To initialize an array of 1000 zeros:

  ```
  vals = [0 for _ in range(0, 1000)]
  ```

- A list can also be initialized using the list() function: vals = list(), will initialize vals as an empty list.

Python: Lists

Appending, inserting and concatenating

you can add a value to the end of the list using the append method.

```
a = [1, 2, 3]
a.append(5)
# a is now equal to [1, 2, 3, 5]
```

You can insert a value at a specific place in the list using the insert method. list.insert(i, x), will insert x before the i-th element in the list.

```
a = ["a", "b", "c", "d"]
a.insert(2, "derp")
# a is now ["a", "b", "derp", "c", "d"]
```

To stick one list to the end of another one, you can use the + operator:

```
a = [3, 4, 5]
b = [0, 1, 2]
c = a + b
# c is now [3, 4, 5, 0, 1, 2]

a.append(b)
# a is now [3, 4, 5, [0, 1, 2]]
```

Python: Lists

Length

To get the length of a list, you can use the len function:

```
a = [1, 2, 3, 4]
b = len(a)
# b is now 4
```

Sublists

In order to get a sublist of a list you can use the `:` operator inside square brackets.

In general `a[x:y]` will return the sublist of `a` starting at index `x` and ending at index `y`. If you omit `x` then you will get a sublist from the start of the list until the index `y`. If you omit `y`, then you will get the sublist that start at position x and ends at the end of the list.

```
a = [4, 5, 6, 7, 8]
b = a[1:]
# b is now [5, 6, 7, 8]

c = a[:3]
# c is now [4, 5, 6]

d = a[2:4]
# d is now [6, 7]
```

Let’s Look at an Example

`string0.c`

`capitalize2.c`

Python: Printing

Instead of using `printf`, you use the `print` function in Python.

C

```
int a = 7;
char b = "x";


printf("%d\n, a);
printf("%c\n, b);
```

Python

```
a = 7
b = "x"
print(a)
print(b)
```

Python: Printing

No more “\n”!

The print method automatically adds a newline at the end.

If you don’t want a new line:

```
print("a string", end="")
```

C

```
int a = 7;
char c = 'x';

printf("%d, %c\n", a, c);
```

python

```
a = 7
c = "x"

# Method 1
print(a, end = "")
print(", ", end = "")
print(c)

# Method 2
# Here we use the + operator to concatenate strings
and the str function to convert other variables to
strings.
print(str(a) + ", " + str(c))

# Method 3
print("{}, {}".format(a, c))
```

Python: Tuples

A new kind of data type.

They can hold multiple values.

They are ordered,immutable data - you cannot change the values in a tuple once assigned.

Tuples are declared using parentheses.

```
a = ("meaning of life", 42)
# a[0] will return "meaning of life", a[1] will return
42
```

Tuples can be easily unpacked when iterating over a list of tuples as follows:

```
constants = [
    ("meaning of life", 4.2),
    ("pi", 3.14),
]

for name, val in constants:
    print("the value of {} is {}".format(name, val))
```

Python: Dictionaries

- Dictionaries are effectively the equivalent of a hash table in C.
- Alternatively, you can think of a dictionary like a list that you can index into using keywords, rather than numerical indices.
- Dictionaries consist of key-value pairs, where the keys are integers or strings, and the values are anything (including other dictionaries, lists, or tuples)
- Dictionaries are created by assigning a set of key-value pairs in curly braces, each set separated by commas, to a variable.
- Dictionaries are like structures where the contents ARE mutable.

Python: Dictionaries

```
weather = {
    "England": "Rainy",
    "California": "Warm",
    "Florida": "Humid",
    "Estonia": "Cold"
}

# This will print out "humid"
print(weather["Florida"])

# This will change england's weather to windy
weather["England"] = "windy"
```

You can iterate over the **keys** of a dictionary using a simple for loop:

```
for place in weather:
    print(place)
```

This will print out all of the places from the dictionary from the previous example. If you also want the values stored in the dictionaries, you can use the following:

```
for place, status in weather.items():
    print("The weather in {} is {}".format(place,
status))
```

Python: Functions

- Functions are introduced with the def keyword.
- Functions have names and parameter lists, just like in C.
- Python files are interpreted, not compiled, which means they are read top to bottom, left to right.
- Code does not necessarily, but can be, bound up in a main() function, though that requires special extra syntax.
- Functions do not require a prototype, but do need to be defined before they are called.
- Python functions can return multiple values if need be, and may also return tuples, lists, and dictionaries.

Python: Functions

Function are declared using the `def` keyword.

C

```
int square(int a) {
    return a*a;
}
```

python

```
def square(a):
    return a**2
```

Note: In Python you can use the ** operator to exponentiate values.

In Python functions don't have prototypes, but do need to be defined before they are called

Python doesn't have a main function. You can simply start writing code in a `.py` file and it will get run. However if you still want a main function, then you can do the following:

```
def main():
    # do stuff

# this part is important to make sure main gets
executed:
if __name__ == "__main__":
    main()
```

Let’s Look at an Example

`positive.c`

`cough4.c`

Python: Classes and Objects

- Python is an object oriented language! Objects are similar to structures in C in that they have fields (called **properties**). Additionally they have **methods** which are functions that are inherently part of that object, and may only be called directly by those objects.
- You define the methods and properties of an object inside of a class.
- Classes are created using the `class` keyword. Class names conventionally start with a capital letter.
- At a minimum, a class must contain a method called `__init__`, which sets the initial values of properties in the object.
- All methods of classes must include the `self` parameter as their first parameter, which is a reference to the object that is invoking the method.
    - When calling a method in a program, however, the `self` parameter is omitted (it is assumed apply to the object that is invoking the method in the first place).

Python: Classes

In Python, classes are declared using the `class` keyword.

Every class can have a __`init`__ method, that defines how an object of this class should be initialized.

Note the __ before and after the name of the method. In Python you surround a method name with double underscores, to indicate that it should be considered a private method (that shouldn't be called directly).

Every method in a class, must minimally take `self` as a parameter. This allows the object to manipulate values within in. When calling a method, the `self` parameter is omitted as it is assumed that it's the object itself.

Class names in Python start with an uppercase by convention.

It's good style in Python to keep class declarations in separate files.

student.py:

```
class Student():

    def __init__(self, name):
        self.name = name
        self.gradeSum = 0
        self.numGrades = 0

    def getGpa(self):
        return self.gradeSum / self.numGrades

    def addGrade(self, grade):
        if grade == "A":
            self.gradeSum += 4
            self.numGrades += 1
        elif grade == "B":
            self.gradeSum += 3
            self.numGrades += 1
        elif grade == "C":
            self.gradeSum += 2
            self.numGrades += 1
        else:
            print("Invalid grade")

    def getName(self):
        return self.name
```

grades.py:

```
from student import Student

tim = Student("Tim")
lisa = Student("Lisa")

tim.addGrade("A")
tim.addGrade("B")
tim.addGrade("B")
tim.addGrade("A")

tim.addGrade("E")
tim.addGrade("P")

lisa.addGrade("C")
lisa.addGrade("B")
lisa.addGrade("A")
lisa.addGrade("A")
lisa.addGrade("A")
lisa.addGrade("A")

print("{}'s GPA is: {}".format(tim.getName(), tim.getGpa()))
print("{}'s GPA is: {}".format(lisa.getName(),
lisa.getGpa()))
```

Python: Misc.

- Instead of using #include, in Python you use the import keyword e.g., import CS50
- There is no ++ operator. Use += 1, for instance.
- No need for ;
- // is for integer division
- # is for comments

Python: Running

To run a Python file, write `python file.py`, where `file.py` is a placeholder for the name of your file.

Flask

- In addition to command line uses, Python can also be used to write basic web applications.
- HTML is used to build websites, but website written in pure HTML suffer a serious limitation.
- Incorporating Python can make our code so much more flexible.
- Flask is a web framework that's very lightweight, to make this process particularly easy.
- Thanks to Flask, it's very easy to write a simple dynamic web application.

Flask

```
<html>
      <head>
            <title>
                  Current Time
            </title>
      </head>
      <body>
            The current date and time is October
24 2017 11:01
      </body>
</html>
```

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def time():
    now = datetime.now(timezone('America/New_York'))
    return f"The current date and time is {now}."
```

Flask

- We need only import the Flask module to get Flask functionality.
- By default, the file Flask is looking for will be called `application.py`.

- It’s rather easy to get started using Flask within CS50 IDE.

```
from flask import Flask
```

- After importing the Flask module, we need to initiate a Flask application.

```
app = Flask(__name__)
```

- From there, we need only write functions to define the behavior of our application.

```
def time():
```

Flask

- Flask typically works by associating function we write with particular *routes*, or URLs. By pairing functions to URLs can we obtain differing behaviors

```
from flask import Flask
from datetime import datetime
from pytz import timezone

app = Flask(__name__)

@app.route("/")
def index():
    return "You are at the index page!"

@app.route("/sample")
def sample():
    return "You are on the sample page!"
```

Resources

- Python’s official documentation (Python 3)
- The Python Language Reference
- The Python Standard Library
- The Python Tutorial
- Flask
- Jinja (Python Templating Engine)

Pset6

- Find similarities between two files (e.g., common lines, sentences, or substrings)
- Display a form (index.html) where user can select two files and compare them

Watch Brian’s walkthroughs! Find out what tools are already available for you!