

CS50 Section

Week 6

Attendance Sign In: tinyurl.com/pythoncs50rw

Agenda

- Week 5: Remaining Pset / General Questions
- Week 6: Python

Week 5 Debrief

Week 5 Review - What Questions do You Have?

- Linked Lists
- Hash Tables
- Trees
- Stacks + Queues

Week 6

Python



<https://youtu.be/8xCzjOnfQbw>

INTRODUCING PYTHON

- Python, as a language, is quite a bit newer than C (1991 vs. 1972), but still is heavily inspired by it.
- Python files are written using the .py file extension, and are run via the Python interpreter.
- Variables in Python do not require a type specifier, and do not need to be declared in advance.

INTRODUCING PYTHON

- Python is a **dynamically typed** (types determined at runtime) and **strongly typed** (you can't mix types, such as trying to add an integer and string together) language
 - C is **statically typed** (types explicitly defined by you) and strongly typed
- Python is an **interpreted language** (instructions run line-by-line without compiling)
 - C was a **compiled language** (compiled into object files before you can run it)

INTRODUCING PYTHON

- What are the advantages/disadvantages of an interpreted language?

IMPORTANT CAVEAT



VARIABLES

- Variables in Python do not require a type specifier, and do not need to be declared in advance.

```
x = 54
```

```
phrase = "This is CS50"
```

VARIABLES

- Variables in Python *do* have underlying types, despite the fact that we don't have to explicitly declare them

```
coursenum = 50
```

```
coursename = "Introduction to Computer Science I"
```

- When we reassign a variable in Python, it's like we “rip off the label” and point to another container (meaning types can change)


VARIABLE TYPES

- We have a number of types we can choose from:
 - number
 - string
 - tuple
 - list
 - dictionary
 - set

VARIABLE TYPES

- We have a number of types we can choose from:

- number
- string
- tuple
- list
- dictionary
- set



Notice that we don't have to specify between `int`, `double`, `float`, etc.

VARIABLE TYPES

- We have a number of types we can choose from:

- number


- string

- tuple

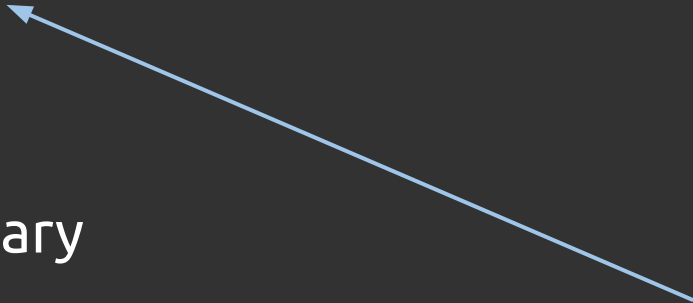
- list

- dictionary

- set



Notice that we don't have to specify between `int`, `double`, `float`, etc.



Notice that we don't have an array of characters anymore!

TUPLES

- The most basic data structure in Python

```
t = (1, 2, "apple", 4.5)
```

- *Can* contain elements of different types
- Is immutable—You cannot increase/decrease it's size
 - Perhaps closest to an array in C
- Notice we use parentheses to create them

TUPLES

- Tuples allow for **unpacking**, in which you split up their values and assign them to different variables:

```
coordinate = (3, 2, 7)
x, y, z = coordinate
```

LISTS

- Similar to tuples, but are *not* immutable

```
l = [1, 2, "apple", 4.5]
```

- *Can* contain elements of different types
- You can increase/decrease their size as you wish
- Notice we use square brackets to create them

WORKING WITH LISTS

OPERATION	DESCRIPTION
<code>list.append(x)</code>	Appends an item <code>x</code>
<code>list.extend(x,y,z)</code>	Extends a list with items <code>x</code> , <code>y</code> , <code>z</code>
<code>list.insert(i, x)</code>	Inserts <code>x</code> at position <code>i</code>
<code>list.remove(x)</code>	Removes first item in the list whose value is equal to <code>x</code>
<code>del list[i:k]</code>	Removes elements from <code>i</code> to <code>k</code>

SETS

- Sets are unordered and contain no duplicate elements

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
```

- *Can* contain elements of different types
- You can increase/decrease their size as you wish
- Notice we use curly brackets to create them

WORKING WITH SETS

OPERATION	DESCRIPTION
<code>set.union(s)</code>	Returns all elements in <code>set</code> and/or <code>s</code>
<code>set.intersection(s)</code>	Returns all elements common to <code>set</code> and <code>s</code>
<code>set.difference(s)</code>	Returns all elements in <code>set</code> , but not <code>s</code>
<code>set.add(x)</code>	Adds element <code>x</code> to the set
<code>set.remove(x)</code>	Removes element <code>x</code> from the set

DICTIONARIES

- Dictionaries follow the key-value pair structure

```
tel = {'jack': 4098, 'sape': 4139}
```

- *Can* contain elements of different types
- You can increase/decrease their size as you wish
- Notice we use curly brackets to create them
- You access a specific value by indicating the key it belongs to: `tel['jack']` returns 4098

WORKING WITH DICTIONARIES

OPERATION

DESCRIPTION

```
dict["existing_key"] = <val>
```

Sets `existing_key` to `<val>`

```
dict["new_key"] = <val>
```

Adds `new_key` to dictionary and sets it equal to `<val>`

```
del dict["existing_key"]
```

Deletes the key-value pair for `existing_key` from the dictionary

FINDING THE LENGTH OF DATA STRUCTURES

- You can use `len(x)` to find the length of any data structure in Python

DATA STRUCTURES EXERCISES

<http://bit.ly/31E4Arz>

DATA STRUCTURES EXERCISES (SOLUTIONS)

<http://bit.ly/2RbGQGB>

CONDITIONALS

C

```
int x = get_int();
if (x < 0)
{
    printf("x is negative\n");
}
else if (x > 0)
{
    printf("x is positive\n");
}
else
{
    printf("x is zero\n");
}
```

PYTHON

```
x = cs50.get_int()
if x < 0:
    print("x is negative")
elif x > 0:
    print("x is positive")
else:
    print("x is zero")
```

SOME KEY DIFFERENCES WITH CONDITIONALS

- We use the keywords `and`, `or`, `not` instead of `&&`, `||`, `!`
- Use `elif` instead of `else if`
- No equivalent of the `switch` statement in Python
- Body code introduced with a `:` instead of `{ }`
 - **Must be indented and whitespace matters!**

THE WHILE LOOP

C

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

PYTHON

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

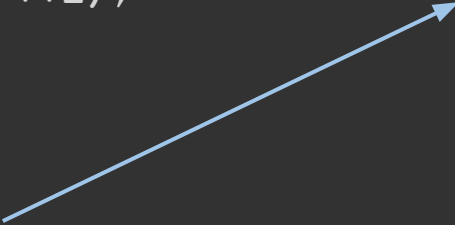
THE WHILE LOOP

C

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

PYTHON

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



Notice the use of
indentation and the :
symbol

THE FOR LOOP

C

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

PYTHON

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

THE FOR LOOP

- The `while` loop is quite similar to its C counterpart, but the `for` loop is much more robust and powerful in Python than in C
- `for` loops in Python don't actually iterate over indices, but instead iterate over sequences

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

`range()` returns a sequence from 0 to 101, counting up by 2 each time:

(0, 2, 4, ..., 100)

FUNCTIONS

- Just like in C, we have functions which have an input and output
- However, functions are modified to fit the “Pythonic” style:
 - You don’t have to specify types for the parameter list
 - You don’t have to specify a return type (including for `void` functions)
 - Introduce functions with the `def` keyword

FUNCTIONS

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

```
base = cs50.get_float()  
print(square(base))
```

FUNCTIONS

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


Indentation and whitespace
matters! *Are you catching
onto a theme?*

```
base = cs50.get_float()  
print(square(base))
```

FUNCTIONS

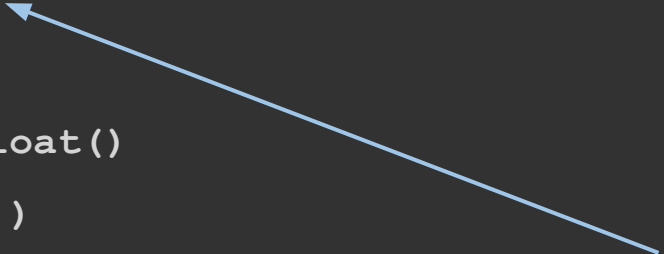
```
def square(x):  
    return x ** 2
```

Indentation and whitespace
matters!



```
base = cs50.get_float()  
print(square(base))
```

Notice because we have
simplified types (just
number), we don't have to
handle different number
types individually



OBJECTS

- In C, we could create our own new “data types” by establishing structures:

```
struct address
{
    char name[50];
    char street[100];
    char city[50];
    char state[20];
    int pin;
};
```

OBJECTS

- Python has this functionality as well, but through **objects**
- Objects in Python can have properties and methods
 - Just like in C, **methods** are the fields of data we want to store
 - However, new to Python, we can also give objects **methods** which are functions inherently part of that object
- We used a **struct** to define the “template” of a structure in C, and we use a **class** to define an object’s “template” in Python

OBJECTS

```
class Student():
    def __init__(self, name, year="Freshman"):
        self.name = name
        self.year = year

    def endYear(self):
        if self.year == "Freshman":
            self.year = "Sophomore"
        elif self.year == "Sophomore":
            self.year = "Junior"
        elif self.year == "Junior":
            self.year = "Senior"
        else:
            self.year = "Alum"

    def info(self):
        print(f"{self.name} is a {self.year}.")
```

OBJECTS

```
class Student():  
    def __init__(self, name, year="Freshman"):  
        self.name = name  
        self.year = year
```

Use the `class` keyword to define a new class

The `__init__` function is our object constructor

```
    def endYear(self):  
        if self.year == "Freshman":  
            self.year = "Sophomore"  
        elif self.year == "Sophomore":  
            self.year = "Junior"  
        elif self.year == "Junior":  
            self.year = "Senior"  
        else:  
            self.year = "Alum"
```

We can include as many functions as we want to define

```
    def info(self):  
        print(f"{self.name} is a {self.year}.")
```


OBJECTS

```
# create two new students, one is a freshman
```

```
maria = Student("Maria", "Senior")
```

```
newkid = Student("John Harvard")
```

← Declaring new objects

```
# everyone graduates at the end of the year
```

```
maria.endYear()
```

```
newkid.endYear()
```

← Calling the methods of an object

```
# new years, now!
```

```
maria.info()
```

```
newkid.info()
```

OBJECTS

- When we create a struct in C, we can declare it empty (just reserving space for it in memory) or initialize it with values
 - Similarly, we can create a new *instance* of an object in Python with or without initialized values
 - Python looks for `__init__` in our class definition and starts there when you declare a new object

OBJECTS

- We use the `self` keyword so that our methods can operate on our object
 - You are effectively “passing” the object to its own methods to perform some set of operations on it
- You don’t need to use `self` when calling the methods of an object; This is done automatically for you
 - But `self` is always needed as the first parameter of every method you define in the class (if you want your methods to do something to the object itself)

HANDS ON PRACTICE

<http://bit.ly/2RcgleF>

HANDS ON PRACTICE - SOLUTIONS

<http://bit.ly/2R4NDSa>

USING LIBRARIES

- To include files akin to what you did in C, use Python's import!

```
import cs50
```

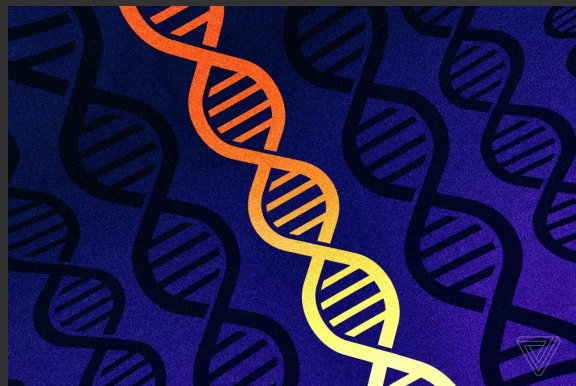
- Then you can use the functions inside of CS50's *module*.
- To run Python programs using your IDE's interpreter, simply type:

```
python <file>
```

PROBLEM SET 6 PREVIEW

Due Sun 10/27 @ 11:59pm

- Hello
- Mario (Less / More Comfy)
- One of: Cash / Credit
- Readability
- DNA



Section Feedback: <https://tinyurl.com/cs50rwfeedback>