IntroPython_Spring_2016 Documentation

Release 0.1

Brian McMahan

June 17, 2016

CHAPTER 1

Course Description

Computing technology has integrated itself into every aspect of our lives. In this course, we will tour through one of the most popular programming languages: Python. Python is used at companies like Google, Microsoft, Facebook, Amazon, and Apple to accomplish a huge variety of tasks. Its versatility, similarity to the English language, and large community support make it one of the best programming languages for learning.

This course will cover the basics of problem solving with Python. We will cover standard data types, loops, conditional statements, functions, and classes. Students will not only learn the basics of syntax, but also how to solve problems with programming. The course will prepare students to move forward to more complex topics at Heroes Academy, or dive into self-taught studies at home.

CHAPTER 2

How to Browse This Document

This document is intended to be a companion to the Introduction to Python course taught at Heroes Academy. For more information about Heroes Academy, please visit it here.

Below and to the left you will find the sections of this document. Each week there will be exercises to complete at home, as well as supplementary materials for further understanding and learning. Python has a rich suite of tools for problem solving and carrying out computational tasks. We will cover the fundamentals without delving too deeply into the more sophisticated features that require extra study.

2.1 Course Information

2.1.1 What is HEROES Academy?

HEROES Academy is an intellectually stimulating environment where students' personal growth is maximized by accelerated learning and critical thinking. Our students enjoy the opportunity to study advanced topics in classrooms that move at an accelerated pace.

2.1.2 When does this course meet?

The Intro to Python course will meet from 11:30 AM to 1:30 PM on the following Sundays:

- April 10, 17, 24
- May 1, 8, 15, 22
- June 5, 12, 19

2.1.3 How do I register for this course?

This course has already begun, but new courses are started at regular intervals! The list of courses are listed on the HEROES website. If you have any questions about the process, you can check out the HEROES Frequently Asked Questions.

2.1.4 What are the expectations of this course?

I expect that...

1. You will ask questions when you do not get something.

2. You will keep up with the work.
3. You will fail fast:
   - Failing is good
   - We learn when we fail
   - We only find bugs when code fails; we rarely hunt for bugs when code is working
4. You will not copy and paste code from the internet
   - You are only cheating yourself.
   - It won't bother me if you do it, but you will not learn the material.
5. You will try the homework at least once and email me with solutions or questions by Wednesday

2.1.5 How do I contact you?

You can reach me anytime at tmeo@njgifted.org

2.2 Installing Python

2.2.1 Python Distribution

There are several ways to get Python. My recommended way is the Anaconda distribution. It includes both Python and a bunch of other things packaged with it that make it super useful.

Instructions for downloading Anaconda Python:

- Click the link above.
- If you use a Mac, look at the section titled "Anaconda for OS X," and click on "MAC OS X 64-BIT GRAPHICAL INSTALLER" under the "Python 3.5" section.
- If you use a Windows computer, in the section titled "Anaconda for Windows," click either "WINDOWS 64-BIT GRAPHICAL INSTALLER" or "WINDOWS 32-BIT GRAPHICAL INSTALLER" under the "Python 3.5" section.
- On most Windows machines, you can tell if it's a 64-bit or 32-bit system by right-clicking on the Windows logo and selecting "System." The line labeled "System Type" should say either 64-bit or 32-bit. If you're having trouble with this, simply email me and I'll help you out!
- Once you click the button, an installer file will be downloaded to your computer. When it finishes downloading, run the installer file.
- Follow along with the prompts, and select "Register Anaconda as my default Python 3.5" if you're using the Windows installer.
- At the end of the installation wizard, you're done! Anaconda, and Python, are installed.

2.2.2 An Editor

There are many good editors and IDEs (Integrated Development Environments). As you're just beginning to learn how to use Python, it's a good idea to use a simplistic, lightweight development environment. PyCharm and Sublime Text are both good choices for starting out. They have nice, clean appearances, highlight your code to make it easier to read, and are easy to jump in and start coding right away.

Instructions for downloading PyCharm:

- Click the link above.
- Click "Download" under the "Community" section.
- An installer file will be downloaded to your computer. When it finishes downloading, run the installer file.
- Follow along with the installer, and select "add .py extension" if you see the option
- At the end of the installation wizard, you're done! PyCharm is now installed.

Other than those two, GitHub has an editor that is very comparable to Sublime Text. It is called Atom.

2.3 General Resources

2.3.1 Online Books

- How to think like a Computer Scientist
- How to think like a Computer Scientist: Interactive Edition
- A collection of links to Python guides

2.3.2 Debugging Help

- 16 common Python runtime errors for Beginners

2.3.3 Interactive Coding Websites

These are some excellent websites that let you code and compete online:

- Hackerrank
- Codewars
- CodinGame

2.3.4 Online Code Environments

There are plenty of website out there that will let you test out Python code online. Trinkets is a great resource that we'll use a lot during this course.

C9 is a more powerful environment which students can also use if they're looking for a more advanced experience.

2.4 Week 1: Hello World

2.4.1 Summary

This was our first lesson! We introduced ourselves and covered classroom basics (ask questions if you don't understand, learning is more important than being right, etc.). We discussed what computers are, what input and output are (input = you feed information to the computer, output = the computer returns some information to you), and what

computer programming actually is (giving instructions to a computer). Python is just one language you can use to give a computer instructions.

Values are data - things like 25, “Hello”, and 3.14159. Variables are just containers that hold that data. Each variable you use in code gets its own name - it’s like an envelope that you label so you remember what’s inside of it. You make variables in Python using the “assignment” operator, which is the equals sign (=). Here are some examples:

x = 5

my_text = “Hello, World!”

num3 = 3333.333

text_number = “500”

(Remember - you can tell if a variable is a String if it’s surrounded by ‘’ or “”)

There are 4 main types of data in Python:

- Integers (numbers with no decimal place)
- Floats (numbers with a decimal place)
- Strings (text, surrounded by quotes)
- Booleans (True or False)

We learned three commands:

- print(), which prints out whatever you put in the parentheses
- type(), which evaluates the type (integer, float, string, boolean) of whatever is in the parentheses
- len(), which evaluates the length of whatever is in the parentheses. For example, len(“Hello!”) = 6

We also previewed some of Week 2’s material, mostly just the following simple mathematical operators:

“+” addition, 3 + 5 = 8

“-” subtraction, 10.1 - 6 = 4.1

“*” multiplication, 2 * 2 = 4

“/” division, 11 / 2 = 5.5

There are also two special math operators. The first is “//”, or floor division. This acts like remainder division, but leaves off the remainder. So, 13 // 5 = 2, and 4 // 100 = 0. And “%” is modulo, which acts like remainder division but only says the remainder. So, 5 % 3 = 2, 100 % 50 = 0, 7 % 10 = 7, etc.

We went over these toward the end of class, so we’ll review them at the beginning of Week 2.

2.4.2 Homework

The homework for this week is to get Python, and PyCharm, installed and working on your home computer. Instructions on how to do so are located in the “Installing Python” section on the left.

Once you’ve installed Python (the Anaconda version, as shown in the “Installing Python” section), you’ll automatically get a program on your computer called QTConsole. You can search for this either by using the Windows key (on a Windows machine) or by searching for it in the Finder (on a Mac). This is the console we used in class - try out some code on your own and see what you can do!

We won’t be using PyCharm for a little while, so just see if you can install and open the program - don’t worry about making anything with it.

And remember: if you have any difficulty, email me at tmeo@njgifted.org with questions!

2.4.3 Extra Resources

None

2.4.4 Lecture Slides

2.5 Week 2: New Operations and Input

2.5.1 Summary

We covered the following topics in Week 2 of class:

- Inserting a new line in a String
- Concatenating (combining) Strings
- Repeating a String
- Indexing Strings
- Slicing Strings
- Math Shortcuts
- Converting between types
- User Input

Inserting a new line in a String

You can use \n in the middle of a String to make a new line. For example, the String "Hello, \n World!" will print like this:

```
Hello,
World!
```

You can also use \t in the middle of a String to make an indent. "Hello, \t World!" will print like this:

```
Hello,     World!
```

Concatenating Strings

You can combine Strings using the + sign.

Example:

```
str1 = "Hello"
str2 = "World!"
str3 = str1 + str2
print(str3)
```

This will print out "HelloWorld!"

Repeating a String

You can repeat Strings using the * sign

Example:

```
str1 = "bogdan"
str2 = str1 * 3
print(str2)
```

This will print out "bogdanbogdanbogan"

Indexing Strings

You can get one character from a String using square brackets, []. Inside the square brackets, put the index of the character you want to get. In a String, the first character starts at index 0, and goes up from there.

For example: If str = "computer", then:

- str[0] is "c"
- str[1] is "o"
- str[2] is "m"

...and so on.

You can put -1 in the brackets to get the last letter of a String too.

- str[-1] is "r"
- str[-2] is "e"

etc.

Remember, every character gets its own index – even numbers, symbols, and spaces!

Slicing Strings

By getting a slice of a String, you can get multiple characters all at once. Use square brackets for this too. Inside the brackets, you first put the starting index, then a colon, and then the ending index.

For example:

```
str = "fantastic!"
print(str[0:3])
```

This will give you "fan". It starts at 0, and stops just before the character at position 3. So, you get the letters at positions 0, 1, and 2.

Some more examples:

- str[1:4] is "ant"
- str[0:2] is "fa"
- str[3:7] is "tast"

...and so on. If you leave out the first number, the slice will start at the beginning of the String.

- For example: str[:5] is "fanta"

If you leave out the second number, the slice will go until the end of the String.

- For example: str[2:] is "ntastic!"

Math shortcuts

Let's say you're writing code and have a variable x = 5. What if you want to increase x by 10? You could do this:

```
x = x + 10
```

Python gives you a shortcut way to write this:

```
x += 10
```

`x += 10` is a way of telling Python, "just increase x by 10." You can also do `x -= 10` to decrease x by 10.

You can use this shortcut with the following math signs:

- +=
- -=
- *=
- **=
- /=
- %=

Converting between types

In Python, variables all have a type. If you do `my_number = 5.1234`, then the variable `my_number` has type Float (because it's a number with a decimal point).

In Python, sometimes you can convert variables to be a different type. For example, remember that there are two kinds of numbers in Python: int (no decimal) and float (with a decimal). You can convert from one to the other:

```
my_float = 5.1234
other_number = int(my_float)
print(other_number)
```

This will print out 5. When you convert a float to an int, Python simply chops off the decimal part.

Or:

```
my_int = 10
some_float = float(my_int)
print(my_int)
```

This will print out 10.0 (Python just adds a decimal point when you convert an int to a float).

If you have a String that is just a number, for example, var1 = "100", you can convert that to an int or float!

```
var2 = int(var1)
var3 = float(var1)
```

One note of caution: if you have a String variable like `my_string_variable = "50.3"`, you can't directly convert it to an Int (because it has a decimal point). If you want it to be an Int, you'd have to first convert it to a Float, and then to an Int.

Finally, you can convert just about anything to a String.

```
my_num = 505.606
some_text = str(my_num)
print(some_text)
```

This will print out "505.606" – a String!

User Input

The last thing we learned in Week 2 was how to get user input. This is where you ask the user to type in a value, and can use that value in your code! You do it with the input() function. Inside the parentheses, you put a String, which is the message that the user will see.

Here's a quick example. Type the following code into the Python shell:

```
user_name = input("Please type in your name: ")
```

If you type that code in and press enter, it will display the message, "Please type in your name: " and wait for a response. Type something in (any name will do) and press enter. Then type the following code:

```
print(user_name)
```

It should print back out whatever you typed in! The name you typed is saved in the variable `user_name`, so you can treat it like any normal String.

Maybe you want to print out how many letters are in your name:

```
name_length = len(user_name)
print(name_length)
```

. . . and so on.

Quick note: whenever you get user input, the computer assumes it's a String. So in the example above, `user_name` is a String. Even if the user types in a number, you get it as a String first. You can convert it to a number using the int() or float() functions we learned.

2.5.2 In-Class and Homework Exercises

There are three different exercises for homework this week. They are all ".py" files, so you should open them using PyCharm!

The homework files are located on the course Github page. To download a file, right-click on its name and select "Save link as..." then select where you want to save it.

Once you've downloaded the homework files, open PyCharm on your computer. Click "File", then click "Open", and select one of the homework files. (remember, they end in .py) We haven't used PyCharm in class yet, so don't worry - you're just using it to view the homework. As you can see, PyCharm makes reading code way easier than a basic text editor!

Once the file is open, you'll be able to read my instructions (they're always at the top of the file) and go from there.

- formulas.py - Nice and simple! I've written a few formulas for you to try out (like the ones we did in class - area of a circle, etc). See if you can write the code for them in Python!
- strings_practice.py - This one is also pretty quick.
- harder_formulas.py - This file has a few word problems that you can solve using Python! These are a bit harder, and it's fine if you can't get through them. They're a bonus challenge.

Try to give these a shot by Wednesday, and send me an email with answers and/or questions. You can reach me at tmeo@njgifted.org. If you have trouble getting PyCharm to work, or can't download the files, you can ask me about that too. I'm happy to answer any questions!

I've included the lecture slides below in case you forget how to do anything we talked about in class. Good luck!!

2.5.3 Extra Resources

Coming Monday!

2.5.4 Lecture Slides

2.6 Week 3: PyCharm, Booleans, and If Statements

2.6.1 Summary

We started class with a review of Week 02's material. If you want to review again, you can either use the lecture slides, or look at the Week 02 summary.

We also talked about PyCharm, a program you can use to write code. PyCharm is different than using the IPython shell; in the shell, you type one line of code at a time, hit Enter, and see what happens. PyCharm lets you write as many lines of code as you want, and then have the program execute them all, one after another. This is super useful if you're writing a long or complicated program. It's also a big time-saver, since you can run the same code over and over again without having to re-type it each time!

PyCharm also lets you **save** Python files, which is useful if you start to write code in class, and maybe want to bring it home with you or work on another computer. You can save a Python file (it will end in .py) and e-mail it to yourself or copy it onto a USB drive.

To practice using PyCharm, we wrote a program that lets you type in a word and get the Pig Latin translation. If you want some extra practice with PyCharm, try writing this program yourself at home!

Next we talked about Booleans. Booleans are variables that can have a value of `True` or `False`. You can set Boolean variables in code with something like `x = True`, or you can use **comparison operators.**

These are the comparison operators we discussed:

- < less than
- > greater than
- <= less than or equal to
- >= greater than or equal to
- == equal to (remember, in Python, "equal to" uses two equals signs, because one equals sign is just used for making a variable)
- != not equal to

Comparison operators compare the values of two different variables, and will evaluate to either True or False. For example, `5 > 3` will evaluate to `True`, but `10 == 9` will evaluate to `False`. You can use these to make Boolean variables as well.

Booleans can also be combined using the `and` and `or` keywords. If `x` and `y` are Booleans, the expression `x and y` will only be `True` if both `x` and `y` are `True`. `x or y` will only be `True` if at least one of them is `True`. And of course, `not x` will just be the opposite of `x`.

We practiced evaluating Booleans using cards and complex conditions (suite == hearts and not number <= 5).

Finally, we introduced the If Statement. An If Statement is comprised of two ingredients: a condition (which must be a Boolean), and some code. Python checks if the condition is True; if it is, the code will be executed. But if the condition is False, Python will just ignore the code and move on.

If statements kind of resemble a paragraph - the condition goes at the top, and the accompanying code is all indented by 4 spaces.

```
if <condition>:
        do some code
        do some more code
back to normal code
```

The computer knows when the If Statement paragraph ends because the indentation stops. That's all!

We also took a sneak preview at the `Turtles` module, but that will be covered in next week's lesson.

2.6.2 In-Class and Homework Exercises

I want to make sure everyone's PyCharm is working properly, since a few people mentioned they'd had some trouble. Now that we've used PyCharm in class, try using it at home, following the guide on the Week 3 Github page. The filename is "Using PyCharm.pdf", and you can either view it directly on GitHub, or download it.

Email me back once you've tried it out, and let me know either if it worked, or if you ran into a problem! If you had a problem, just describe what it was, maybe with a screenshot of the issue. We'll work to get it sorted out, and then proceed from there.

Good luck!

2.6.3 Extra Resources

2.6.4 Lecture Slides

2.7 Week 4: Turtles, Elif-Else, and For Loops

2.7.1 Summary

This week we learned more complex types of If Statements: The If-Else, and If-Elif-Else structures.

It helps to think of the three of them like this:

- An If statement gives the computer one option: if `<condition>` is True, then do something. That's all.
- An If-Else statement gives the computer two options: if `<condition>` is True, then do something. If `<condition>` if False, do some other thing!
- An If-Elif-Else statement gives the computer several options, where you can say "Check all of these conditions until you find one that's True."

Each kind of statement is indented in the same way - with 4 spaces. Here's an example of each:

If Statement:

```
if x == 5:
        print("x is 5!")
```

If-Else Statement:

```
if x == "Penny":
        print("Your name is Penny!")
else:
        print("Looks like your name isn't Penny!")
```

If-Elif-Else Statement:

```
if age == 50:
        print("You're really old!")
elif age == 20:
        print("You're kind of young!")
elif age == 10:
        print("You're a kid!")
else:
        print("I wonder how old you are?")
```

You can put in however many "elif" portions you want. The computer will just go through each of the conditions, one after another, until it finds one that's True. Then, it will skip the rest of the paragraph. And if none of the conditions are True, it will do whatever is written under the "else" section.

We talked about the concept of Objects in Python. A Turtle is an example of a Python Object. You can make a Turtle, show a turtle on-screen, move it around, print out information about it, etc.

More specifically, Objects have two qualities: **functions,** which are things that the object can *do*, and *properties,* which are pieces of information that describe an object.

Every different type of Object has its own unique set of functions and properties. For example, a Turtle Object has the functions `forward()`, `left()`, `right()`, etc. Notice how functions always end in parentheses!

We'll talk more about Objects later - and eventually you'll learn how to write your own! - but for this week, I just wanted to introduce the concept. Objects are **things**, functions are **actions**, and properties are **details** about an Object.

We spent quite a while working with Turtles and doing various activities - drawing shapes and words, using different colors, etc. A link to the Turtles Cheat Sheet I handed out is included in the "Extra Resources" section below, so look there if you want to try some out on your own!

The last thing we learned about is the `for` loop. `for` loops are great - they use indented lines to form a 'paragraph' (kind of like If statements!) and let you run the code in that paragraph over and over again, as many times as you want!

Say you wanted to print someone's name 10 times (kind of a ridiculous example). The loop would look like this:

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

That's it! If you execute this code in Python (easier to type it into PyCharm than the shell), it will print out "Cinder" ten times in a row.

Breaking it down:

- `for` is a special keyword - when Python sees it, it knows we'll be repeating some code
- `i` is just a variable, just like `x` or `username`
- `range(10)` is the list of all numbers from 0 to 9

In the above For loop, Python will repeated the indented code 10 times, and each time, `i` will take a new value.

- First time through: `i` is `0`
- Second time through: `i` is `1`
- Third time through: `i` is `2`

etc.

So you can also do something like this:

```
for i in range(5):
        print(i)
```

This will print 0, 1, 2, 3, and 4, because the code will execute 5 times, and each time, `i` has a different value!

For loops can be tricky to wrap your head around. The best thing to do is to use the above two examples, copy them into PyCharm, and verify that they work. Then try changing the number in range(), and also change around what happens in the indented text. The best way to practice new coding techniques is to try it yourself

2.7.2 Homework

No homework this week! Be ready for next week - we'll be reviewing a lot!

2.7.3 Extra Resources

Turtle Cheat Sheet

2.7.4 Lecture Slides

2.8 Week 5: Collections and Loops

2.8.1 Summary

We covered a lot of material this week. This recap will be important for getting the homework done!

Topics:

- Review of `for` loops
- Review of If-Elif-Else structure

See the page for Week 04 for a review of those two topics.

Collections

Collections are variable types that can hold more than one value - not just an int or a String, but a *sequence* of values. We learned about three types: Lists, Tuples, and Dictionaries.

Lists in Python are simply that - a linear, ordered bunch of values. Lists can have ints, Strings, booleans, etc., for their members. You can make an empty list like this:

```
grocery_list = list()
```

Or, you can make one like this:

```
grocery_list = []
```

Finally, you can make a list that already has items in it:

```
grocery_list = ["bread", "milk", "beans"]
```

You can get items from a list using the same syntax as indexing and slicing strings (see Week 02 for a refresher). For example, `grocery_list[0]` will return the String "bread", and `grocery_list[1:]` will return ["milk", "beans"]. Notice how when you return just one item, the type is whatever the item was - a String, int, etc. But if you get multiple elements, it's just a shorter List.

- Reassign List items: `grocery_list[1] = "bacon"`
- Add an item to the end of a List: `grocery_list.append("butter")`
- Delete a particular item: `del grocery_list[1]`
- Get the length of a list: `len(grocery_list)`

Dictionaries in Python work like real-world dictionaries; instead of organizing items by number, each item gets a "key", and you can look up items by their "key." Dictionaries are great for when you want to store information and don't care about how it's ordered - you just want to be able to look up specific entries by name.

To make a blank dictionary and add items to it:

```
my_dict = {}
my_dict["first entry"] = "This is the first entry!"
my_dict["second entry"] = "This is the second entry!"
```

Then, `print(my_dict["first entry"])` will print "This is the first entry!"

The values in a Dictionary can be Strings, Ints, Booleans, anything! The keys can be Strings, Ints, or Tuples.

Tuples in Python are very much like Lists. The main difference is that the items in a tuple can't be changed once they've been set. Tuples are useful for when you have a set of values that you know won't change, and don't want to allow the program to change.

To make a Tuple:

```
num_tuple = (0, 1, 2)
```

If you try `num_tuple[1] = 5`, Python will complain.

While Loops

A `while` loop is another kind of loop - it works differently than a `for` loop. `while` loops have two parts: a `<condition>`, and a body of code. When Python reaches a `while` loop, it checks to see if `<condition>` is True. If it is, the code in the code body will be executed.

Once that's finished, Python will again check `<condition>`. If it's True, the code will execute again, and again, and again...This continues until `<condition>` is False. So be careful - a `while` loop can continue forever if `<condition>` never becomes False!

Syntax of a `while` loop:

```
x = 5
while x < 10:
        print("The loop is still going!")
print("Looks like the loop finished!")
```

The above is an example of an **infinite loop**. x never gets changed, so it'll *always* be less than 10. The final line will never be reached!

Bonus

Finally, we learned a cool trick with `for` loops and Collections (list, dictionary, etc.) All of these are examples of **iterables** - objects in Python that you can loop over by taking the first item, and then the next, and the next, etc.

And you can use any iterable in a for loop - it doesn't just have to be `range(x)`! Check out the following example:

```
grocery_list = ["olive oil", "eggs", "ham", "celery"]
for item in grocery_list:
        print("Remember to buy: ")
print("That's it!")
```

The above code will output:

```
Remember to buy: olive oil
Remember to buy: eggs
Remember to buy: ham
Remember to buy: celery
That's it!
```

And that brings us to the end of the lesson!

2.8.2 Homework

For homework this week, you'll write 3 different files in PyCharm. Email each to me when you've finished them! You can either email the file itself, or just copy and paste your code into the email. My address is tmeo@njgifted.org

Remember, ask questions if you have any difficulty! These three assignments are a good way to tell if you've really got the basics down, which is important this far into the class.

Practicing with Lists

Make a list that uses different types of variables for its items (use ints, booleans, and strings). It should be at least 5 items long.

After you make the list, ask the user to enter a number. Print out the **type** of the item at that index in the list. (Do you remember how to do this? If you've forgotten, try using Google!)

A word of caution - like we saw in class, if the index number is too high, the program will run into an error! Use an `if/else` statement to check if the number they gave is too high. If it is, just print a message telling the user they should have picked a smaller number, and move on.

Finally, use a `for` loop to print out each item in the list individually.

Practicing with Dictionaries

Pick 5 random words, either from a physical dictionary or using Google. Make a Python dictionary whose keys are the words, and whose values are the definitions (you can keep the definitions short).

At the end of the program, ask the user to enter one of the words in your dictionary. Print out the definition of the word they chose, using you Python dictionary.

Something to think about: What happens if the user types in a word that isn't in your dictionary yet? Try it out - we'll see how to deal with this in a future lesson...

Practicing with While Loops

Have a variable x that starts at 0, and another to represent an upper limit. Repeatedly ask the user for a number, and increase x by that amount.

Keep asking for numbers until x has risen above the upper limit you chose. At the end, print out what the final value of x is - and tell the user whether it's even or odd (hint: you'll have to use one of the math operators...)

2.8.3 Extra Resources

2.8.4 Lecture Slides

2.9 Week 6: Basic Functions

2.9.1 Summary

This week we talked about **functions** - what they are, what're used for, and how to write our own.

Function Basics

So what is a function? The short answer is, it's a bunch of lines of code that you set aside - kind of like a special *paragraph* - and give a name to. Then, anywhere else in your code, you can use that same name to execute the function's code, without having to type it all out again. Just by using the name, the computer will know what code you're talking about.

Here's an example: Say you wrote some code that prints a bunch of sentences in a particular order, like this:

```
print("First sentence\n")
print("Second sentence\n")
print("Final sentence!\n")
```

If you wanted to write this code as a **function**, it would look like this:

```
def three_sentences():
        print("First sentence\n")
        print("Second sentence\n")
        print("Final sentence!\n")
```

Things to note: - `def` (define) is a keyword that tells Python you're about to write a function - The next word is the name of the function (you choose this - it can be whatever you like), followed by parentheses (these also indicate to Python that it's a function) - The line ends with a colon, just like loops and `if` statements. As always, the contents of the function - its "paragraph" - are indented by 4 spaces

Defining Functions and Calling Functions

The code above just **defines** the function called `three_sentences`. None of the code will actually be executed; we;re just letting the computer know that in the future, if we say `three_sentences()`, we're talking about this paragraph.

After you've **defined** the function like we did above, you can **call** it anywhere in your code. Calling a function is the same as executing a function. You can call a function simply by writing the function name, followed by parentheses. For example, look at this code block:

```
def three_sentences():
        print("First sentence\n")
        print("Second sentence\n")
        print("Final sentence!\n")

print("OK, let's call the function!\n")

three_sentences()
```

The final line of code actually calls the function. Once a function has been defined, you can call it as many times as you want! You can also define as many functions as you want in a single program.

Function Arguments

The function above is really simple - you just call it, and it does something. Some functions, like `print()`, are different - you *need* to put something in the parentheses, because it's expecting something to be in the parentheses.

The thing you put in the parentheses is called an **argument**. That's just another word for the input of a function.

We can write functions that take arguments too. For example, let's say you wanted to write a function where, when somebody calls it, they need to put a name (probably a String) in the parentheses, and the function will print out a greeting for that particular person. It would probably look something like this:

```
def greeting(name):
        print("Hello there, " + name + "!\n")
```

To write a function that takes an **argument** in its parentheses, you simply write a *variable name* inside the parentheses like I did with the `name` variable above. Then you can use that variable in the function!

Now, when you call the `greeting()` function, you need to put something in the parentheses, like this:

```
greeting("Penny")
greeting("Emerald")
greeting("Cortana")
```

If you try to just call `greeting()`, Python will complain, because when you *defined* the function, you told it to expect something in the parentheses.

Python doesn't check which type of variable an argument is, so even if you're expecting a String, someone could still type `greeting(5.127849)` without crashing the program.

You can even have more than one argument in a function! Check out the example below:

```
def add_two_numbers(num1, num2):
        sum = num1 + num2
        print(sum)
        print("\n")
```

If you put that at the top of your program, now you can call it to get the sum of any two numbers! For example, try `add_two_numbers(0, 5)`, `add_two_numbers(100, -56)`, and `add_two_numbers(.0456, .55903)`. s you can see, multiple arguments are just separated by commas, both when **defining** a function, and also **calling** a function.

Scope

We briefly discussed this in class - just a little warning to keep in mind when working with functions. In our `add_two_numbers(num1, num2)` function above, `num1` and `num2` are the arguments that the functions expects. They're both variables that we can use within that function's *paragraph*.

However, outside the paragraph, if you try to reference `num1` and `num2`, Python will complain that it doesn't know what variables you're talking about. This is because `num1` and `num2` **only** exist within the function's paragraph.

So, for example:

```
def add_two_numbers(num1, num2):
        sum = num1 + num2
        print(sum)
        print("\n")
```

```
print("Let's sum two numbers!")
add_two_numbers(1, 2)
print(num1)
```

...will crash, because of the last line. We'll talk more about scope later on.

We finished up by experimenting with turtles and writing functions. Check the Extra Resources section after tomorrow to see some examples!

2.9.2 Homework

I have 3 exercises I'd like you to complete in PyCharm this week. They might seem simple, but it's important to get the basics of writing and using your own functions in code early on, so it's worth the practice.

1. Simply write a program that defines the `add_two_numbers` function from above. Call the function with 5 different pairs of numbers. Then try the following two experiments: What happens if you call it with a pair of Strings? What happens if you call it with one Int and one Boolean?
2. Write a program that defines the following functions.
   - `difference_between_two_numbers(num1, num2)`, which subtracts one number from the other and prints the result
   - `multiply_two_numbers(num1, num2)`, which multiplies the two numbers and prints the result
   - `compare_two_numbers(num1, num2)`, which prints which of the two numbers is bigger (hint: you'll need an if statement for this one!)
   - Then, after you've defined these functions, call each one once to demonstrate that it works.
3. Write any function using the `turtle` module. It can be as simple as you like (for example, `draw_straight_line`), as long as it 1) takes a **turtle** as an **argument**, and 2) makes that turtle do something. Be creative and challenge yourself!

Remember to email me with questions and answers as tmeo*njgifted.org. I'm always happy to help!

Good luck!

2.9.3 Extra Resources

When I finish putting it together, I'll post some of the `turtle` code that we wrote using functions

2.9.4 Lecture Slides

2.10 Week 07: Advanced Functions and Basic Classes

2.10.1 Summary

Important Note: No class next week! Don't come in on Sunday, May 29th. Week 08 resumes on Sunday, June 5th.

This was our final week of all-new material! We covered 2 topics: functions that have **return** statements, and an intro to Python **classes**. Short summary below - more information can be found in the lecture slides.

Functions with Return Statements

Last week, we only talked about functions that take input arguments and print things. But what if we wanted to write a function that returns a value you can put in a variable?

The answer is a **return** statement. At the end of a function, use a return statement to have the function spit out a particular value. Then, when you call that function, you can put the returned value in some variable.

Here's an example:

```
def addition_func(x, y):
        result = x + y
        return result
```

Then, if you want to call this function, you can do this:

```
the_sum = addition_func(10, 15)
```

We call the function, and put the return value into the `the_sum` variable.

Python Classes

Finally, we learned the basics of defining and using our own custom-made object classes. The basic idea behind **defining** a class is that you're writing a recipe for a particular type of object. you can think of it like this: if you have a room full of chairs, each of those chairs is a chair object, but "chair" would be the name of the class.

After you've defined a class (written your recipe), you can use it to make copies of your custom-made object in code. The Lecture Slides have example code in case you forget!

See the "Extra Resources" section for examples. In short, the proper syntax is this:

```
class <Class_Name>:
        property_a = value_a
        property_b = value_b
        property_c = value_c

        def some_class_function(self)
                <code>
                <code>
                <code>
```

Remember, classes have two very important features in Python: **properties**, which are details about the object that describe it, and **functions**, which are things that the object can **do**.

For example, a `Dog` object in Python might have the properties `name`, `age`, `height`, etc., and functions like `run(self)`, `bark(self)`, and `fetch(self)`. Remember that when you're defining functions inside an object, you need to make the first argument (the first thing in the parentheses) the keyword `self`, which tells Python, "this function belongs to this object type."

Similarly, inside of a class's function, if you want to reference one of that class's properties, you also need to use the `self` keyword. So, in the `bark(self)` function for a dog, if you wanted to print its name, it would look like this:

```
def bark(self)
        print("Hello! My name is " + self.name)
```

Don't forget the `self` keyword!

2.10.2 Extra Resources

These are the classes you guys wrote in this week's lesson. Use them as an example!

The `Clown` class:

```
class Clown:
        name = "Trisha Perthen Ferdletate Torhalimil Ludwig Sonyetta Paghetti Careyeep"
        funniness = 1
        makeup = "a lot"


        def laugh (self) :
                if self.makeup == "a lot":
                        print ("BWAH-HA-HA-HA-HA-HA-HA! I'M GOING TO MAKE YOU WISH YOU NEVER CAME TO
                elif self.makeup == "only a little":
                        print ("HAHA! WHAT will I have to do to make THIS little kid CRYYYY?")
                else:
                        print ("I still have to think about how to torture this little kid in their h

        def smile (self) :
                if self.funniness == 1:
                        print ("YOU ARE LUCKY! TODAY I ONLY HAVE A FEEBLE SMILE WATTAGE. :)")
                elif self.funniness == 5:
                        print ("HA HA HA! YOU ARE NOT IN LUCK! MY SMILE IS CHARGED FULLY! GOOD LUCK I
                else:
                        print ("MY AWESOMELY AWESOME NICENESS IS COMING INTO PLAY, BECAUSE I WILL GIV
```

The `Tornado` class:

```
class Tornado:
        rank=0
        speed=72
        duration= 1

        def set_rank(self, rank_num):
                self.rank = rank_num
                if self.rank==0:
                        self.speed=72
                elif self.rank==1:
                        self.speed=112
                elif self.rank==2:
                        self.speed=157
                elif self.rank==3:
                        self.speed=205
                elif self.rank==4:
                        self.speed=260
                elif self.rank==5:
                        self.speed=318
                else:
                        print('You screwed up! If you KNEW about tornadoes, you\'d KNOW that they go
                        quit()

        def shout(self):
                # if self.rank==0:
                print("The tornado has rank " + str(self.rank), "we are going to see speeds up to " +
```

The `Computer_Virus` class:

```
class Computer_Virus:
        type = "Type A"
        power = 900000
        destruction_level = 900000000

        def destruction(self):

                if self.destruction_level == 1000000:
                        print("erasing everything but your hairline because you don't have one")
                elif self.destruction_level == 900000000:
                        print("..........................................................." * 723)
                else:
                        print("jkasergtervfasdgfhygawebvfrn dsavgsvfheawhfaehdnhZbn,vfabndvfyukwevgba
```

The `Door` class:

```
class Door:
        color = "brown"
        height = 7
        number = 5

        def a(self):
                if self.height == "7":
                        print("good")
                if self.height >= "7":
                        print("go away")
                if self.height <= "7":
                        print("get out")
```

2.10.3 Homework

Because we have 2 weeks until next class, try to do this assignment before next Sunday. Then on Sunday, I'll post another assignment for the following week.

For homework this week, you'll be writing another class. You can pick any object you want to write a class for - however, you need to include the following requirements:

1. The class should have at least 3 properties (remember, properties are just internal variables)
2. The properties should include at least one Boolean, at least one String, and at least one Int
3. The class should have at least 2 internal functions
4. At least 1 internal function needs to somehow use a property of the class (remember to use the `self` keyword!)
5. At least 1 internal function needs to return a value

6. At least 1 internal function needs to take an input argument 6. The functions and properties should be meaningfully named (for example, no names like "x," "a," or "var")

Then, once you've defined the class, write some code that does the following:

1. Make an object of that class
2. Change one or two of the properties of the class, so they aren't just the default values
3. Call the class's functions

This is mostly just a review of what we covered this week and last week. Next Sunday, the assignment will be a little more complex.

Remember to send me an email at tmeo@njgifted.org if you have any questions. Good luck!!

2.10.4 Lecture Slides

2.11 Week 8: Advanced Classes

2.11.1 Summary

This lesson was split up into two segments:

- First, we learned about the `__init__` function in classes
- Second, we spent most of the lesson reviewing everything we've learned so far about Python.

The `__init__` function is one of Python's special functions - this is indicated by the double underscore (`__`) on either side of the function name. `init` is a keyword (like `print` or `if`) and Python already knows what it's used for.

When you write your own class, sometimes it's helpful to have a kind of setup function that runs whenever you make a new copy of the class. For example, if you write the `Door` class we've been using as an example, you might want the `Door` to print out "Hello!" the first time someone makes it. And, every new `Door` that gets made will also say "Hello!"

This is what the `__init__` function is for: it's a special function that runs once every time an object of that type (in our example, `Door`) is made.

So, for example:

```
class Door:
  def __init__(self):
    print("Hello!")

first_door = Door()
second_door = Door()
```

The code above will print out "Hello!" twice - once for `first_door`, and again for `second_door`.

That's an example of an `__init__` function that doesn't take any arguments. Usually, this isn't the case - because `__init__` is a setup function, you want the user to provide certain information about the object when they make it.

Here's an example:

```
class Door:
  def __init__(self, in_name, in_height):
    self.name = in_name
    self.height = in_height
    print("Hello! My name is " + self.name)

first_door = Door("Gerald", 10)
second_door = Door("Geraldina", 12)
```

In this code, when a `Door` object is created, it takes two arguments: the name, and the height. These arguments are then used for setting up the Door object (i.e., they set up the properties `self.name` and `self.height`)

After that, we did a big review - all the questions and answers are included in the lecture slides, so you can use them if you want to brush up on old material.

2.11.2 Homework

No homework this week - just review what we've learned to prepare for your final project!

2.11.3 Lecture Slides

2.12 Week 09: Application

2.12.1 Summary

We spent the entire lesson working on final projects. In case you forget, the project description and list of requirements are available in the lecture slides below. We'll be presenting final projects next week - the first half of the class will be spent wrapping up and debugging the projects, and the second half will be the actual presentations.

We also learned how to do a few new things that could spice up the final project.

- Play .wav sound files in Python
- Generate random numbers
- Move a Turtle object using the arrow keys

Play Sound Files in Python

(this method will only work on Windows computers!)

We covered two methods of doing this. First, if you simply download a sound file and leave it on your computer somewhere, you'll need to give Python the location of the sound file. For example, let's say I'm playing a sound file called "rocket_ship.wav", located in `C:\Users\Public\Downloads` on my computer. To play that, you'd use the following code:

```
import winsound, sys

winsound.PlaySound("C:\\Users\\Public\\Downloads\\rocket_ship.wav", winsound.SND_FILENAME)
```

Let's break down the code above. You import `winsound`, a Python library that can play sounds on a Windows computer, and `sys`, which lets Python perform operations on the file system (such as opening and closing files).

The `winsound.PlaySound` function actually opens and plays the sound file - we give a String for the first argument, which is the location of the sound file - remember to include the file name and extension! (`.wav` in this case) Remember that in order to write a backslash () in a Python string, you need to use a double-backslash. Otherwise, it may think you're trying to use a special String symbol, such as `\n` or `\t`.

For the second argument, we tell Python that that String was just the name of a sound file. That way, the program knows to look in that location, open the file, and play it.

Alternatively, you can put the sound file right in your PyCharm project folder. If you do, you don't need to supply the full path to the file - you can just use the filename (Python will check the project folder for it). The code would look like this:

```
import winsound, sys

winsound.PlaySound("rocket_ship.wav", winsound.SND_FILENAME)
```

This also makes it easier to move your code onto a different computer - you can just copy the whole project folder, sound file included.

Generate Random Numbers

Generating random numbers in Python is easy! Look at the following sample code:

```
import random

x = random.randint(0, 10)
print(x)
```

This code will print a random number from 0 to 10.

Move a Turtle Object Using the Arrow Keys

You can use the following code to set up a Turtle object and move it with the arrow keys:

```
import turtle

bob = turtle.Turtle()

# set the speed at which Bob turns and moves forward/backward
move_speed = 10
turn_speed = 10

# functions that will be called by the arrow keys
def move_turtle_forward():
  bob.forward(move_speed)

def move_turtle_backward():
  bob.backward(move_speed)

def turn_turtle_left():
  bob.left(turn_speed)

def turn_turtle_right():
  bob.right(turn_speed)

# don't have Bob draw a line as he moves about the screen
bob.penup()
bob.speed(0)
bob.home()

# with this code, when the user hits the designated key on the keyboard, the specified function will
screen.onkey(move_turtle_forward, "Up")
screen.onkey(move_turtle_backward, "Down")
screen.onkey(turn_turtle_left, "Left")
screen.onkey(turn_turtle_right, "Right")

# this continually has the program wait for the user to hit keys on the keyboard
screen.listen()

turtle.done()
```

That's it! The explanations for the code are included in its comments.

2.12.2 Homework

Next week, for the final lesson, you'll be presenting your project to the class. For each project, we'll do three things:

- You'll provide a brief overview of your project to the class
- We'll run through the project a few times, testing various input choices and seeing what happens
- We'll all look through the code together to see how it works

For homework, apart from having your project ready (or mostly ready - you will have an hour at the start of class to finish it), you'll need to put together your project overview. I've provided an example in the "Extra Resources" section below. It's really short - just three slide or so, to give us a general idea of what your project is and what it does.

At a minimum, the overview should include:

- What your project is about and what it's supposed to do
- Whether it's complete, or if not, what features are missing
- A few things you learned while working on the project
- One or two things you could do to expand it in the future

Remember to email me with any questions about either your final project or the overview!

2.12.3 Extra Resources

Project Overview Example

2.12.4 Lecture Slides

2.13 Week 10: Goodbye World

2.13.1 Summary

To come after the lesson!

Preview: Final class!

2.13.2 Extra Resources

To come after the lesson!

2.13.3 Lecture Slides

To come after the lesson!

CHAPTER 3

Indices and tables

- genindex
- modindex
- search