IntroPython_Fall_2016 Documentation

Release 0.1

Brian McMahan

Mar 12, 2017

Getting Started

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.

Course Information

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.

When does this course meet?

The Intro to Python course will meet from 11:20 to 1:20 starting October 2nd.

How do I register for this course?

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.

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

How do I contact you?

You can reach me anytime at teacher@njgifted.org

Installing Python

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.

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.

General Resources

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

Debugging Help

- 16 common Python runtime errors for Beginners

Interactive Coding Websites

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

- Hackerrank
- Codewars
- CodinGame

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.

[Week 1] Hello World

Reminder: if you have any difficulty, email me at teacher@njgifted.org with questions! **Failing is good. Failing silently is bad.**

Review

Important Terms

1. Values
2. Variables
3. Input/Output
4. Statements
5. Assignment
6. Expressions
7. Comments
8. Operators

Values

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)

Functions

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.

Homework

Required

1. Get Python installed and working on your home computer. Instructions on how to do so are located in the "Installing Python" section on the left.

   - Open up the interactive shell (iPython console or iPython QT console), play around like we did in class!
   - Use Python like a calculator! Write down the numbers or equation you use and why.

2. Make a cheat sheet for all of the terms listed at the top of the page.

   - Be sure to write down the syntax for assignments and operators.

Optional

1. Make at least one mistake that creates an error. Write it down how you created it. If you can, explain why it happened.

Lecture Slides

[Week 2]: Strings and Input

Summary

Today we covered strings and type conversions. You should work through the problems below and get used to the differences between strings and numbers.

Take Home Exercises

Please see the Week 2 Exercises page for the exercises you should complete. If you would like an extra challenge, solve the problems in harder_formulas.

Review

After this class, you should know or practice all of these topics:

- Inserting a new line in a String
- Concatenating (combining) Strings
- Repeating a String
- Indexing Strings
- Slicing Strings

- Formatting 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!”

Formatting Strings

Formatting strings is necessary if you want to be able to print variables to the shell.

There are a couple different ways of formatting strings. I will cover all three here.

1. With string concatenation

```
animal = "bunny"
adjective = "evil"
noun = "the ruler of the world"

our_sentence = "The "+adjective+" "+animal+" wants to be "+noun"."

print(our_sentence)
```

2. With string formatting

```python
animal = "bunny"
adjective = "evil"
noun = "the ruler of the world"

our_sentence = "The {} {} wants to be {}.".format(adjective, animal, noun)

print(our_sentence)
```

The second way is much preferred because you can have fine grained control over formatting options:

```python
a_number = 3432.34234324233462
print("Not formatted well: {}".format(a_number))
print("Formatted: {:0.3f}".format(a_nubmer))

a_string = "euclid the bunny"
print("without formatting options: {}".format(a_string))
print("with formatting options to right align: {:>50}  [end]".format(a_string))
print("with formatting options to center align: {:^50} [end]".format(a_string))
```

The stuff inside the curly brackets specifies the options. The options start with a colon. Then, if it's a number, you can specify the number of decimal points to have. You need the 'f' for the float.

For strings, '>' aligns to the right, '<' aligns to the left, and '^' aligns to the center. The number directly after that is how wide it should be. It will add spaces to adjust.

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:

```python
x = x + 10
```

Python gives you a shortcut way to write this:

```python
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.

Lecture Slides

Trinkets

[Week 3]: Booleans, If-Elif-Else

Refresh your knowledge from Week 2! and if you finish quickly, an extra challenge.

Exercises

1. In class: Booleans
2. In class: If-Elif-Else Statements
3. Week 3 Takehome Exercises

Terms

You should know the following terms!

1. Boolean values (True and False)
2. Boolean Algebra (and, or, not)
3. Code blocks (4 spaces!)
4. Conditional operators (!=, ==, >, <, >=, <=)
5. Conditional Statements (If, elif, else)

Extra Review

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: `(suit == hearts and not number <= 5)`.

If Statements

`if` statements are comprised of two ingredients: a condition (which must evaluate or 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 the only way it will know!

If-Elif-Else

More complex types of `if` Statements: `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>` is 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.

Lecture Slides

[Week 4]: Turtles and For Loops

Turtles cheat sheet

Agenda

1. Complete the refresher exercises: Week 4 Refresher Exercises.
2. Listen to me tell you about for loops and turtles!
3. Work on the Week 4 exercises: Week 4 Exercises
4. Work on these at home (or in class, if you finish fast enough): Week 4 Takehome Exercises

Review

Terms you should know

1. `for` loops
2. the `loop variable`
3. `range`

From Simple to Complex variables

There are two ideas you should combine in your head. The first is about simple variables. Simple variables have a single type. For example, a simple variable can be an integer or a string.

The other idea you should combine is code robots. We talked about code robots in class. Code robots have a very simple design: take an input, give an output.

Combining these ideas, we can talk about complex variables. Complex variables can have multiple simple variables inside them. They can also be several code robots in one.

Turtles are just this! Turtles can have multiple variables, like color and shape. They can also do multiple things. You can have it go forward or have it turn!

Summary of Turtles

Turtles are created from their factory.

```
import turtle
bob = turtle.Turtle()
```

Then, you can make it move and turn:

```
bob.forward(100)
bob.left(90)
```

There are many things you can do:

```
bob.shape('turtle') # change the shape
bob.stamp() # stamp the shape onto the board
x=100
y=100
bob.goto(x,y) # go to this position
bob.penup() # stop drawing when the turtle moves
bob.pendown() # start drawing again
```

Lecture Slides

[Week 5]: Collections and Loops

Refresher

See this page for the refresher!

Exercises

See this page for some exercises!

Bonus: Dictionary exercises!

Review

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!
```

Random

The random library lets you do randomized events. You must always start with importing it.

For example:

```
import random
# num is short for number
num = random.random()
```

You can do random integers and random choices too:

```
import random
num = random.randint(0,10)

pet_names = ["euclid", "fido", "bob"]
selected_name = random.choice(pet_names)
```

With the `random.randint(start,stop)`, the integer sampled is just like `range`: it will only go UP to the stop number. It will never include it.

Lecture Slides

[Week 6]: Basic Functions

Refresher

Week 6 Refresher

Exercises

1. Functions, part 1
2. Functions, part 2

3. Functions, part 3
4. Functions, part 4

Review

Will be posted after class.

Lecture Slides

[Week 7] Advanced collections and functions

Important point for today!

Keep in mind the **code structure**. What is part of a function? What is part of an if statement? You should always know and keep in mind the **structure**.

Refresher

At the start of class, you will be working through a refresher. There are two parts. Part 1 and Part 2

In class Exercises

After the slides, today we will be working with basic classes. You can find the exercises here.

Lecture Slides

[Week 8] Classes and Projects

Overview

Today, we covered the advanced properties of classes. Specifically, we covered the `def __init__` function and talked about how it can be used to setup an object when it is first created! To practice this, you created an class and object in class.

We talked about your projects and you made a flow-chart to plan what you will be doing. The projects all sounded really awesome and exciting!

Additional Guides

I have placed a few more guides onto the cookbook site (which, if you don't remember, you can get to through the shortcut page I maintain hacs.club). Specifically, I added a couple ways of saving and loading files.

If you would like to know anything more or see detailed examples of something, please email me. The sooner you do this, the better!

At-Home Work

Over the next week, you should be working on your project. There is 2 weeks left until project presentations, so not everything needs to be done, but the more you get done, the better your project can be!

The project should have the following properties:

1. Be text based
2. Get input from the user
3. Use that input to make decisions

The projects you have selected have all followed these criterion, so don't forget that this is the core mission! If you get stuck on something hard, it's also best to scale it back first and get it working. Complexity is much easier if you get a simple thing working first.

The tangible goal for next week is to have a minimal working example of your final project. It should be a very small example of what you want to make bigger. So, it should:

1. Display to the user what the intention of the program is
2. Get input from the usre and then use that input to progress your game, story, or program
3. **Provide at least 1 round of interaction**
    - For the games, this could be 1 action by the user and the computer
    - For the stories, this could be 1 set of choices and/or scenes.
    - For the riddle/encoding, this could be 1 question.

If you want your project to have complex things about the user, try to focus on having only 1 or 2 things for now. The hardest part is getting the code in place to handle everything. And it is easier to do that when the amount of things it is handle is small.

If you want an online coding environment to place your demo, I would recommend either trinket.io or repl.it

Lecture Slides

[Week 9] Project Discussions

This week will be looking at your projects to see how far you've come and what you have left.

We will be discussing your final presentations today and what information should be in it.

You are going to be required to have the following:

> 1. A presentation which you will give to your parents. I've included an example one at the bottom of the page to help you with the framework of it.

Your presentation should...

1. Describe the purpose of your project
2. Outline the logic of your code
3. Explain the types of coding decisions you had to make and why you chose what you did

Presentation Link

You will give a presentation to your parents next time we meet. You will have time at the beginning of class to finish things up, but your presentation is due to me that Friday.

Here is the presentation template:

CHAPTER 3

Indices and tables

- genindex
- modindex
- search