Getting Started, Variables and Simple Types

University of Wyoming COSC 1010

Adapted from: *Python Crash Course 3rd Ed* By Eric Matthes

Getting Started

Getting Started

- We can look at writing your first python program!
- Typically the first program one looks at in any language is "Hello World"
- This program simply outputs "Hello World"
- But, it helps to show basic syntax and output of a program
- As well as how to run it!

Running Snippets of Code

- You can run Python's interpreter in a terminal window (Like the one in Replit!)
- This allows you to run bits of code without saving and running an entire program!
- You can tell when a code snippet is in an interpreter if you have `>>>` prepending a statement
- You can start the interpreter by typing `python3` into a terminal on a system with Python installed
  - Within a repl this is the "shell" option

Running Snippets of Code

```
danny:~/.../week03$ python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello Class")
Hello Class
>>>
```

Running Snippets of Code

- The `>>>` is the "python prompt"
- It indicates you are in the python interpreter and can input python code
- This is done by typing in the code and hitting enter
- Helpful when you need to do quick portions of code, but don't want to write a full file to do so

Running a Hello World Program

- Once you have somewhere that you can write python code in (in our case Replit), you can write and run code!
- Before you get started, comments are an important part of writing a program
- In python comments begin with a #
- Anything after a # will be ignored when the computer runs your program
- Comments can be used to:
  - Put your Name in your code
  - Document what a line of code is doing
  - Make notes to yourself, or people who may view the file
  - Add any additional documentation

Running a Hello World Program

- You should always comment your code
- In ths class your files will always need to begin with:
  - Your name
  - COSC 1010
  - Submission Date
  - Lab Section: [Your lab section]
  - Sources, students worked with online sources used
- So, what would the hello world python program look like?

In [ ]:
```
# Danny Radosevich
# COSC 1010
# 8-23-23
# Lab Section: 10
# Sources used: None

print("Hello World")
```

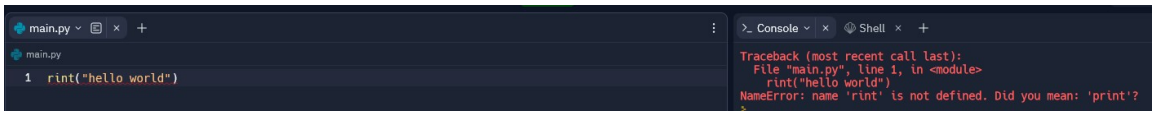
```
Hello World
```

Running a Hello World Program

- Pretty simple right?
- Note, printing from a file is done in the same fashion as we did in the interpreter
- This shows you how to get output from any code you write!
- the output will be sent to the console associated with your program
- In Replit you can run files in a couple different fashions
  - Clicking the `run` button at the top
  - In the **shell** typing `python3 main.py` (or whatever your file is named)
- If you use `run` the output will be sent to **console**
- If you manually run your program in shell, the output will be sent to the **shell**

Troubleshooting

- Writing programs can be difficult!
- If your program doesn't run on the first try that's ok, that doesn't mean it will *never* run
- There are many things to do if your program doesn't run
- If your program doesn't run due to an error python will give you a traceback message



Troubleshooting

- If you need to, take a step away from your computer
  - Clearing your head and coming back does wonders for problem solving
- Start over, if you are unable to find the issue in your code try rewriting it
  - Sometimes rewriting it helps to catch any mistakes made
- Have someone else look over your code
- Ask for hel online (Only for the strong of will)

Variables and Simple Types

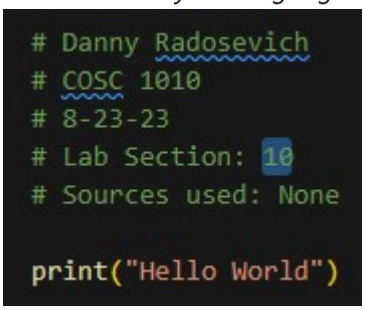
What really happens when you run "Hello World"

- What really happens when running a Hello world program?

- well, python does a lot of the work for you
- Remember the program is simply `print("Hello World")` with the output being `Hello World`
- A python file ends with the extension `.py` , which indicates it is a python file
- This file is run through the python *interpreter*
- The interpreter then reads through the program and determines what each word in the file means
- When it sees `print()` it prints to the screen whatever is inside the parenthesis

What really happens when you run "Hello World"

- When you write a program the editor highlights different parts in different ways
- it recognizes that `print()` is the name of a *function* and displays the word in one color
- Where as the content within the parenthesis would be a different color, as it is not Python code
- This is called *syntax highlighting*



Variables

- We can start talking Variables
- Variables in programming are similar to those in math
- They store a value
- Assigned with the `=` operator
- So `x = 5` we assign the integer value of `5` to be held in our variable `x`
- Unlike in math `x = x + 1` is a valid statement

Variables

- We can add a variable to our Hello Word code
- Previously our code was `print("Hello World")`
- Let's try assigning `"Hello World"` to a variable

- And then print using the variable!

In [ ]:
```
# Danny Radosevich
# COSC 1010
# 8-23-23
# Lab Section: 10
# Sources used: None

message = "Hello World"

print(message)
```

```
Hello World
```

Variables

- We added a *variable* to our code, named `message`
- Every variable is connected to a *value*, the information associated with that variable
- In this case `message` contains the *string* `"Hello World"`
- Adding a variable makes a little more work for the interpreter
  - When it processes the first line, it associates the variable `message` with `"Hello World"`
  - When it reaches the second line of code, it prints the value associated with `message` to the screen
- Now, let's expand the code to print a second message

In [ ]:
```
# Assign "Hello World" to our variable
message = "Hello World"
#Print out the message
print(message)

#Now, we are going to assign a new message to the same variable
message = "Hi there COSC1010"
print(message)
```

```
Hello World
Hi there COSC1010
```

Variables

- Now when the code is run, there are two lines of output
- You can change the value of a variable in your program at any time
- Python will keep track of the current value after each change

Naming and Using Variables

- When using a variable in Python, there are a few rules and guidelines
- Breaking some of these rules will cause errors
- Other rules are in place just to make code easier to read and understand

Naming and Using Variables

- Keep these rules in mind:
  - Variable names contain only letters, numbers, and underscores (Though numbers are rarely used)
    - They must start with a letter or underscore
    - `message_1` and `_message` are valid
    - `1_message` is not
- Spaces are not allowed in variable names
  - use underscores instead!
- Avoid using Python *keywords* and function names as variables
  - These are things like `print`
- Variable names are short and descriptive
- Be careful when using the lowercase letter *l* and uppercase *O* as they may look like 1 and 0

Avoid Name Errors When Using Variables

- Everyone makes mistakes, and most make mistakes daily
- Good programmers may create errors, but they also know how to fix those errors
- When an error occurs Python tries its best to help you
- Let's look at an intentional error, and one that is likely to occur

In [ ]:
```
# assign a message
message = "Go Pokes!"
#now print that message, intentionally misspelling message
print(emssage)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_6549/1780198087.py in <module>
      2 message = "Go Pokes!"
      3 #now print that message, intentionally misspelling message
----> 4 print(emssage)

NameError: name 'emssage' is not defined
```

Avoid Name Errors When Using Variables

- The interpreter provided a traceback, specifically with the error message `NameError: name 'emssage' is not defined`
  - A *traceback* is a record of where the interpreter ran into trouble trying to run the code
- It also tells us the line the error occurred on, in this case `4`
- The interpreter also shows the line
- This allows you as the developer to easily (hopefully) track done the error

Avoid Name Errors When Using Variables

- In this case it was a name error `NameError: name 'emssage' is not defined`
- It reports that the variable being printed `emssage` has not been defined
- A name error usually means a variable's value hasn't been set, or there was a typo
- If it can, Python will try to tell you a possible correct variable name

Variables are Labels

- You can think of a variable as a box that stores a value in it
- This idea can be helpful when you start out
  - But, it isn;t necessarily an accurate way to describe how variables are represented internally in python
- It's better to think of variables as labels that you can assign to values
- You can say a variable *references* a certain value
- Initially this distinction doesn't matter a ton, but it is better to learn it earlier than later

Strings

- Programmers often do gather some kind of data, and then do something useful with it
- So, it helps to classify those different kinds of data
- The first data type we can look at is a *string*
- Strings are simple at first glance, but can be exceptionally helpful
- Strings are a series of characters

Strings

- Anything inside of quotes in Python is a string

- `"this is a string"`
- `'this is also a string'`
- This allows you to have quotations and apostrophes within strings
- `"when they hear that he's 'a-comin', cause the western folks all know, he's a high-falootin', rootin, tootin', son of a gun from ol' Wyomin"`
- `'And then I said "Python is fun!"'`

Changing the Case in a String with Methods

- One of the simplest tasks that you can do with strings is changing the case of the words in a string
- Take this snippet of code:

```
name = "ada lovelace"
print(name.title())
```

- What is this code likely doing?

In [ ]:
```
name = "ada lovelace"
print(name.title())
```

```
Ada Lovelace
```

Changing the Case in a String with Methods

- The variable `name` refers to the lowercase string `"ada lovelace"`
- The *method* `title` appears after the variable in the `print()` call
- A *method* is an action that python can perform on a piece of data
- The dot `.` after `name` in `name.title()` tells python to make the `title()` method act on the variable `name`
- Every method is followed by a set of parenthesis, as methods often need additional information to do their work
  - title however doesn't need any additional information

Changing the Case in a String with Methods

- The `title()` function changes each word to title case, where each word begins with a capitol letter
- This is useful because you will often want to think of a name as a piece of information
  - for example you may want you program to recognize the input values

    - `Ada`
    - `ada`
    - `ADA`
  - And have the program display them all as `Ada`

Changing the Case in a String with Methods

- There are other useful methods for dealing with case as well
- For example you can change teh case to all uppercase `print(name.upper())`
- Or, all lowercase `print(name.lower())`
- The `lower()` method is particularly useful for storing data
- Often you won't want to trust the casing input by a user, so you'll convert them to lower before storing
- Then when you retrieve the data you can apply the correct casing for your needs

In [ ]:
```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

```
ADA LOVELACE
ada lovelace
```

Using Variables in Strings

- Sometimes you'll want to use a variable's value in a string
- For example you may want to use two variables to represent a first and last name
- Then, you want to be able to display those together in a string

In [ ]:
```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(full_name)
```

```
ada lovelace
```

Using Variables in Strings

- To insert a variable's value in to a string, place the letter `f` before the opening quotation mark
- Put braces around the name or names of variables you want to use in the string
- Python will replace each variable with its value when the string is displayed
- This strings are called *f-strings*
  - The *f* is for *format*

Using Variables in Strings

- Python formats the string by replacing the name of any variable in braces with its value
- So for our code snippet `full_name = f"{first_name} {last_name}"` the output becomes
  - `ada lovelace`
- A lot can be done with *f-strings*
- *f-strings* can be used to compose completable messages
- These messages would use the information associated with a variable
- You can utilize string based methods on variables within *f-strings*

In [ ]:
```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
print(f"Hello There, {full_name.title()}")
```

```
Hello There, Ada Lovelace
```

Adding Whitespace to Strings with Tabs or Newlines

- In programming *whitespace* refers to any non printing characters
  - Spaces
  - Tabs
  - End-of-line symbols
- Whitespace can be used to organize output to make it easier to read

Adding Whitespace to Strings with Tabs or Newlines

- Tab characters can be added by inserting `\t` into a string
- Newline characters can be added with `\n`

- As many or as little tabs and newlines can be added to ensure the string is as you want it
- The ability to add tabs and newlines are extremely helpful

In [ ]:
```
print("python")
print("\tpython")

print("--------------")
print("pythonpython")
print("python\npython")
print("--------------")
print("python\n\tpython")
```

```
python
        python
--------------
pythonpython
python
python
--------------
python
        python
```

Stripping Whitespace

- Extra whitespace can be confusing in programs
- To programmers `'python'` and `'python   '` look effectively the same
- To a program they are two different strings thanks to the tailing whitespace
- Python considers the whitespace to be significant unless told otherwise

Stripping Whitespace

- It is important to think about whitespace
- Often two strings will need to be compared to see if they're the same
- An example may be checking usernames against a stored one to determine equivalency
- Extra whitespace can be confusing in more simple applications as well

Stripping Whitespace

- Python make sit easy to eliminate extra whitespace from data that people enter
- Python can look for extra whitespace on the right and left sides of strings
- To ensure there is no whitespace on the right side `.rstrip()` can be used

In [ ]:
```
string_one = "py    "
```

```
string_two = "thon"

print(f"{string_one}{string_two}")
print(f"{string_one.rstrip()}{string_two}")
```

```
py    thon
python
```

Stripping Whitespace

- `string_one` contained extra tailing whitespace
- When `.rstrip()` is applied that whitespace is removed
- It is only removed temporarily, if you asked for the variable `string_one` again it would be back
- To remove whitespace from a string permanently you have to associate the stripped version with the variable name

In [ ]:
```
string_one = "py     "
string_two = "thon"

print(f"{string_one}{string_two}")
print(f"{string_one.rstrip()}{string_two}")

print(f"{string_one}{string_two}")

string_one = string_one.rstrip()
print(f"{string_one}{string_two}")
```

```
py    thon
python
py    thon
python
```

Stripping Whitespace

- To remove the whitespace from the variable it is stripped from the right and reassigned
- Changing the value associated with a variable is often done in programming
- This is how a variable's value can be updated as a program is executed
- White space can also be stripped from the left with `.lstrip()`
- Or from both sides with `.strip()`

In [ ]:
```
string_one = "py"
string_two = "    thon"
print(f"{string_one}{string_two}")
print(f"{string_one}{string_two.lstrip()}")

string_three = "   python    "
print(f"*{string_three}*")
```

```
print(f"*{string_three.strip()}*")
```

```
py    thon
python
*   python   *
*python*
```

Removing Prefixes

- When working with strings another common task is to remove a prefix
- Consider a URL with the common prefix `https://`
- Suppose you want to remove this prefix to just focus on the URL
- Tis can be done with the `.removeprefix()` method

In [ ]:
```
wyoweb = "https://wyoweb.uwyo.edu/"
print(wyoweb)



print(wyoweb.removeprefix("https://"))
```

```
https://wyoweb.uwyo.edu/
wyoweb.uwyo.edu/
```

Removing Prefixes

- To do this you enter the name of the variable followed by a dot and the method name
- In the parenthesis you supply the string of the parenthesis you want to remove
- Like the methods for removing white space `.removeprefix()` leaves the original unchanged
- To do so permanently you can reassign the variable with the stripped string as its value

Avoiding Syntax Errors with Strings

- One kind of error you will often see is a syntax error
- A *syntax error* occurs when Python doesn't recognize a section of your program as valid
- For example if you use an apostrophe inside a single quote it will produce an error
  - `'cause the western folks all know, he's a high-falootin', rootin, tootin', '`
- Python assumes the first quotation to the apostrophe is the string
- So everything following is an error

In [ ]:
```
print(`'cause the western folks all know, he's a high-falootin', rootin, tootin', '
```

```
  File "/tmp/ipykernel_11649/3334084592.py", line 1
    print(`'cause the western folks all know, he's a high-falootin', rootin, tootin
', ``)

^
SyntaxError: unterminated string literal (detected at line 1)
```

Avoiding Syntax Errors with Strings

- To do this correctly you can use double quotes instead
  - `"'cause the western folks all know, he's a high-falootin', rootin, tootin',"`
- Using double quotes to encapsulate the string allows the use of single quotes and apostrophes in the string

Numbers

- Numbers are used often in programming
  - Keeping score in a game
  - Representing data in visualizations
  - Storing information in web applications
- Python treats numbers in several different ways

Integers

- The simplest numbers in Python
- With integers you can
  - Add (+)
  - Subtract (-)
  - Multiply (*)
  - Divide (/)
  - Raise to a power (**)

Integers

- In a terminal session python returns the result of an operation

```
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 3-2
1
>>> 3*3
9
>>> 9/4
2.25
>>> 3**2
9
>>>
```

Integers

- Python supports order of operations too
- Multiple operations can be used in a single expression
- You ca use parenthesis to modify the order of operations
- this allows Python to know the order you want the expression evaluated

In [ ]:
```
x = 2+3*4
print(x)
y = (2+3)*4
print(y)
```

```
14
20
```

Floats

- Python calls any number with a decimal point a float
- The term is used in most programming languages
- It refers to the fact that a decimal point may appear at any place in the number
- Every programming language must be properly designed to manage decimal numbers, so numbers behave accordingly
- Mostly you can use floats without worry

In [ ]:
```
x = 0.1 + 0.1
print(x)

y = 0.2 + 0.2
print(y)

z = 2 * 0.1
print(z)
```

```
0.2
0.4
0.2
```

Floats

- However, there are some circumstances where you may get an arbitrary number of decimal places in an answer
- This happens in all languages, and is of little concern
- Python tries to find a way to represent all the result as precisely as possible
- this is difficult given how computers represent numbers internally

In [ ]:
```
x = 0.1 + 0.2
print(x)

y = 3 * 0.1
print(y)
```

```
0.30000000000000004
0.30000000000000004
```

Integers and Floats

- When you divide any two numbers you will get a float
  - Even if dividing two integers and the result is a whole number
- If you mix an integer and float in any operation you will get a float
- Python defaults to a float in any operation involving a float, regardless of if the output is a whole number

In [ ]:
```
print(4/2)
print(1+2.0)
print(2*3.0)
print(3.0**2)
```

```
2.0
3.0
6.0
9.0
```

Underscores in Numbers

- When writing long numbers you can group digits using underscores
- This makes large numbers more readable
  - `universe_age = 14_000_000_000` When a number that was defined using underscores is printed, only the number is shown

- Python ignores the underscores when storing these kinds of variables
- This applies even ifa grouping of three isn't used
  - `1000 == 1_000 == 10_00`

In [ ]:
```
universe_age = 14_000_000_000
print(universe_age)
```

```
14000000000
```

Multiple Assignments

- You can assign values to more than one variables using just a single line
- This can help shorten your programs
  - While also making them easier to read
- You just need to separate the values and variable names by a comma
- Python will assign each value to irs respective variable as long as the number of items on each side matches

In [ ]:
```
x,y,z = 1,2,3
print(x,y,z)
```

```
1 2 3
```

Constants

- A constant is a variable whose value stays the same throughout program execution
- Python doesn't have a built in constant type
- Python programmers use all capitol letters to indicate a variable should be treated as constant