Python

Methodologies for Data Science

Lars Sorensen - 16:137:603
Fall 2015 - biglars@cs.rutgers.edu

Python Methodologies for Data Sciences

Introduction

Hello World!

Variables - Assignment - Simple Arithmetic

A way to consider simple data

Keywords

Input from a user

Booleans & Conditionals

Boolean Algebra

Conditional constructs/control statements (if-then-else)

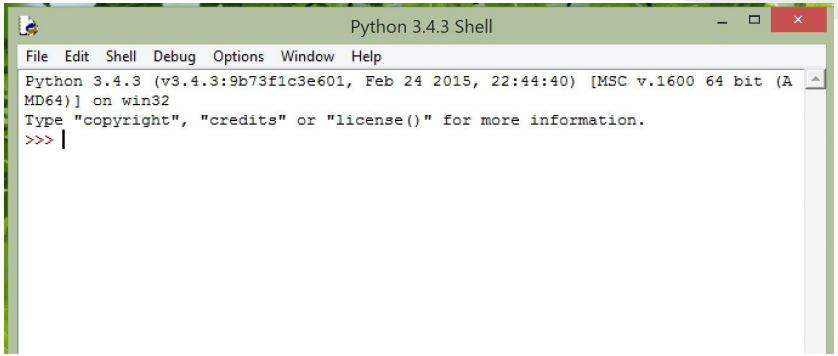
Python Methodologies for Data Sciences

Introduction

Welcome to PMDS. We're going to start slowly but surely and get some of the basics under your belts before we move on to some of the more challenging aspects of computer programming.

Right now I want to make sure that you're using Python 3.4. I will be using IDLE in the slides and YouTube videos as well. Python can be a pain at first because you'll need to have the version you want (3.4 in our case), and it should be for the correct "architecture" for your PC/laptop: either 32- or 64-bit. At the python.org download page you should be able to find an executable that will allow you to load Python 3.4 and IDLE on your system. You can get Python by using the URLs below. Also check out Appendix B of Z (your textbook) on page 479.

https://www.python.org/downloads/
https://wiki.python.org/moin/BeginnersGuide/Download



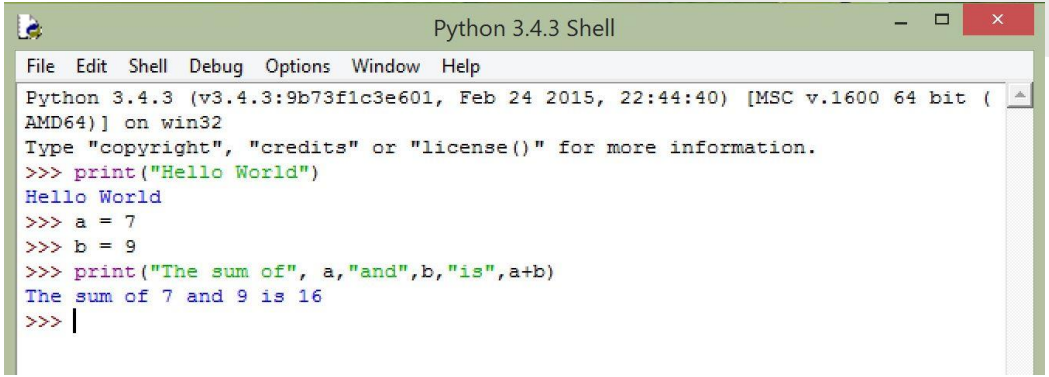
When Python is loaded and IDLE is started, you will see a window like the one above. At first we will use Python in interactive mode, but we will quickly begin saving our work in files. As the course progresses, you'll see that a single program often uses multiple files. All will be revealed in time, though; for now, just make sure Python 3.4 is running on your system.

Python Methodologies for Data Sciences

Introduction

Okay, assuming all is up and running. If this is not the case, then make sure to visit Eric or myself at the CAVE (HILL 252) so we can see what's going on. Don't worry if you didn't run through all the examples and the chaos program from Z chapter one; we'll go through all of that here.

Python lets you use the interactive shell window to just type commands into. This is great when you first begin to tinker with Python but you will quickly discover that you'll want to start combining and saving your code, so we're going to do all of our work with "programs" that we save in files with the ".py" suffix.



The interactive window is still useful, though. That's where your program's output will go, and we will see how it's useful for debugging our programs as well.

Play around with the interactive Python by doing some of the exercises from pg. 23 ex. 1 of Z. You don't have to use the same numbers as he does; play around and do your own thing for while.

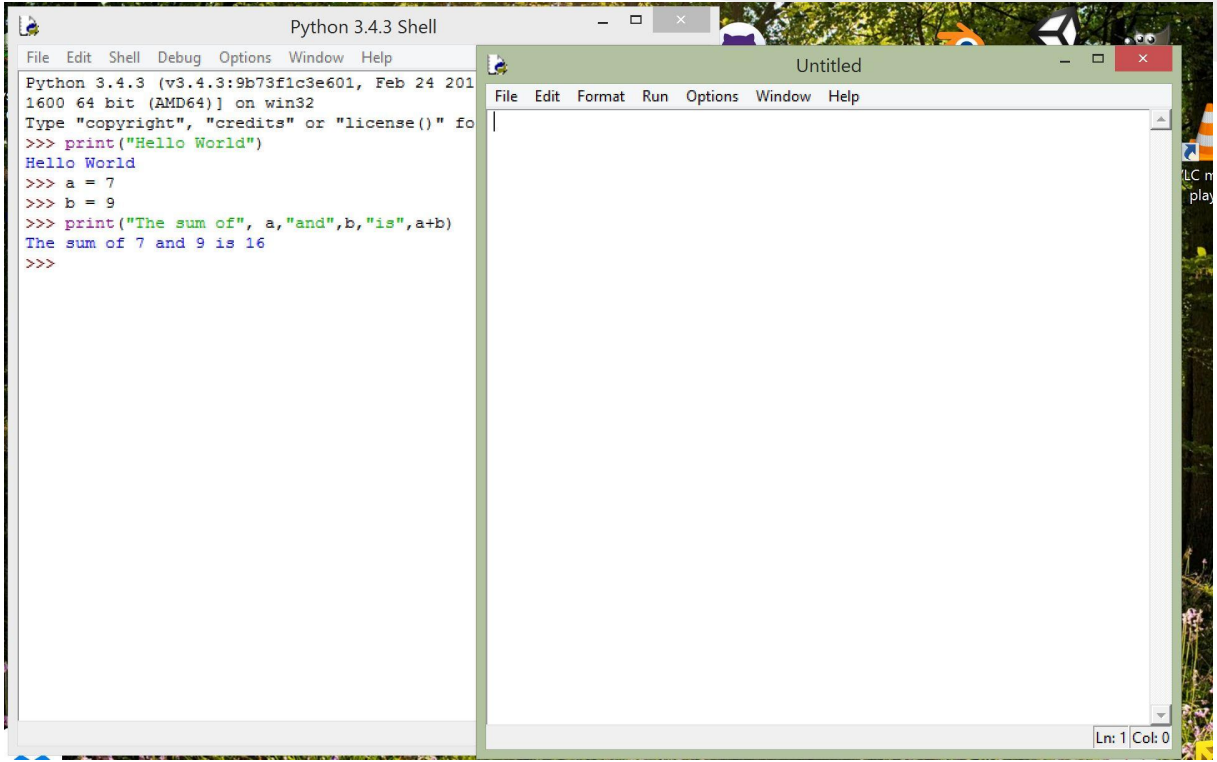
Python Methodologies for Data Sciences

Introduction

In IDLE, click on File and then click on New Window.

A window will open up next to the Python interactive window that says Untitled on the top.

This is where we are going to type our first program. Whenever you learn a new programming language the first program you write is one that prints “Hello World” to the screen. We are not messing with tradition here, so off we go…

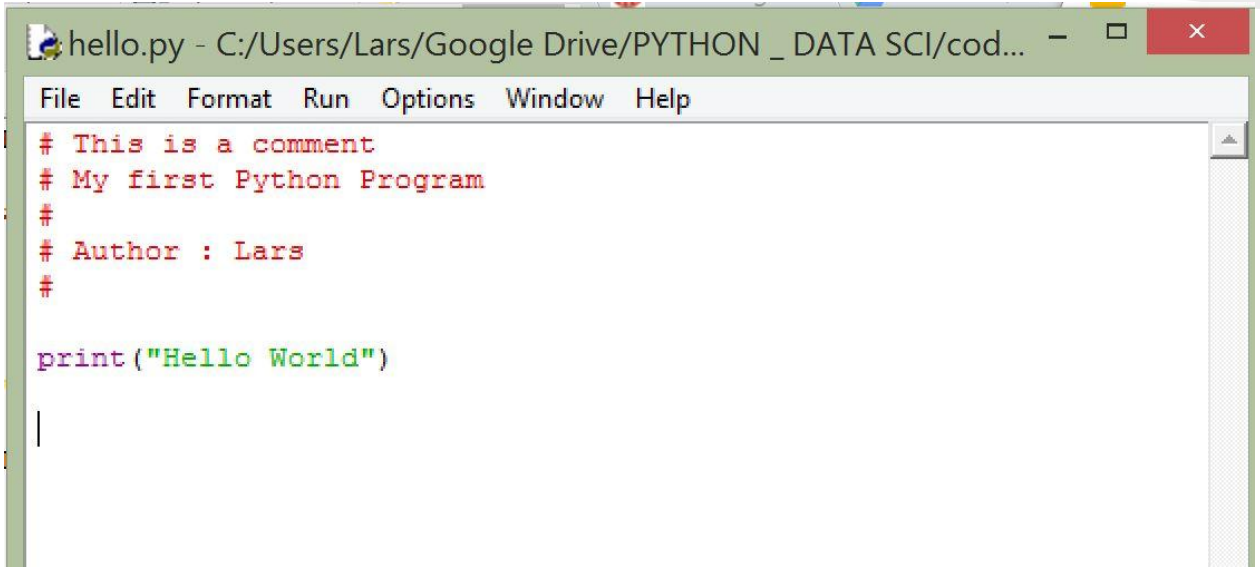


Python Methodologies for Data Sciences

Hello World

In this new window type what you see here (with your name, of course), click File, Save, and save the file in a place where you can keep all your PMDS programs (I keep mine in a google drive so I can share them with you later) and call the program hello.py. The ". py" suffix will tell python and your operating system that this is a python program.

Congratulations. You're a Python programmer now.

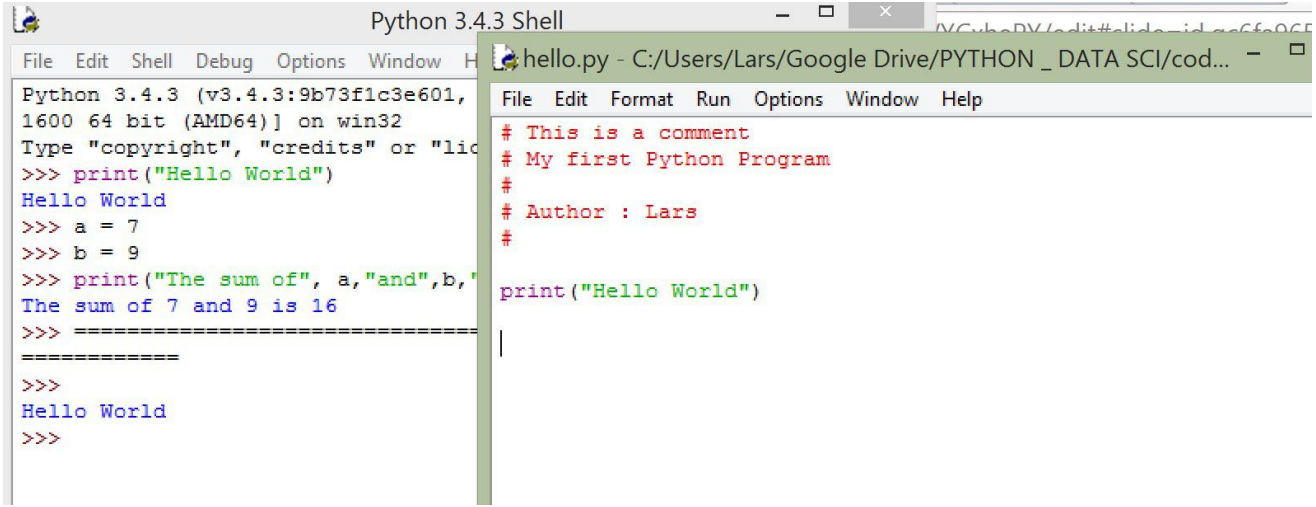


Python Methodologies for Data Sciences

Hello World

Now the only thing left to do is run our program. It's easy. Just click on the Run option and click Run Module (we'll learn about why they call it a module later). Viola! You can see your output in the interactive Python window.

Now, whenever you create a program, a day or two later, you can load that same program and run it again without having to retype anything in an interactive python window.



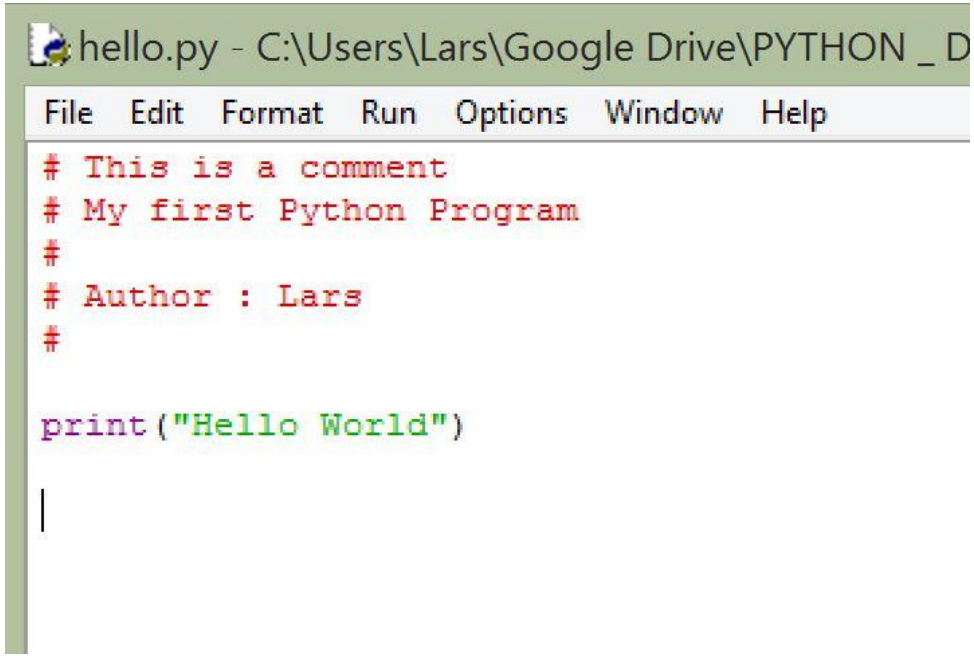
Python Methodologies for Data Sciences

Comments

A quick word on comments.

First, as you can see, we create a comment by typing a number sign and then whatever is after that will not be considered by the python interpreter.

Comments are EXTREMELY important. Code readability is a key facet of programming. All good programs are well commented and explained as they proceed.

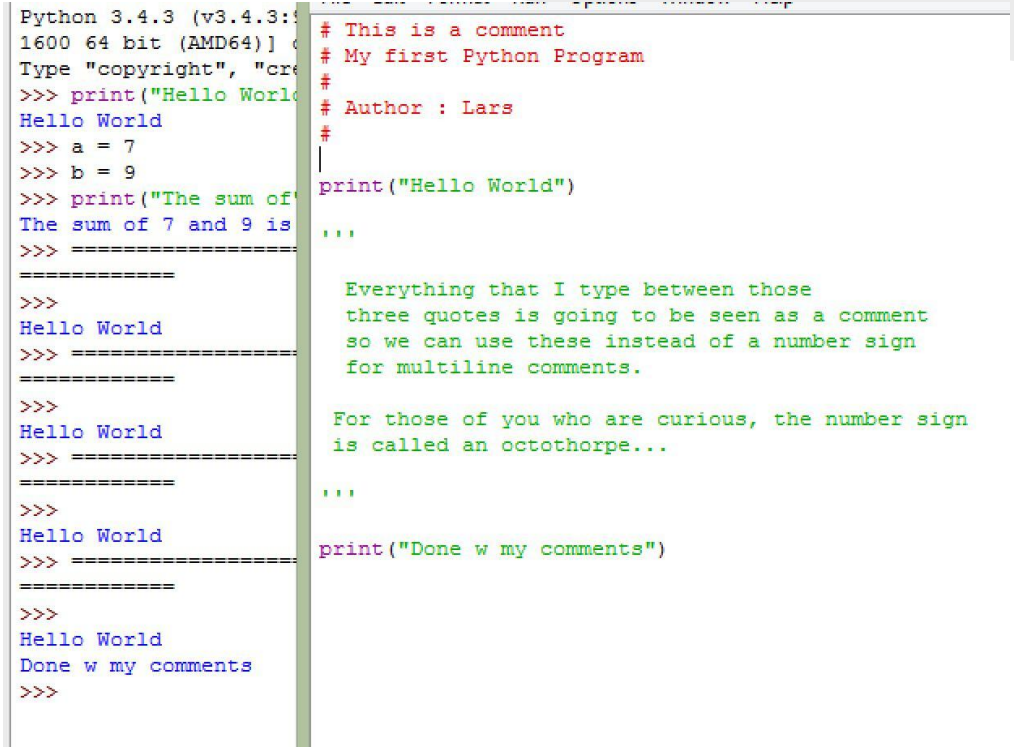


Python Methodologies for Data Sciences

Comments

Another way to create comments is to use three quotes. Once you type the three quotes you can type for many multiple lines and the text will be ignored by the Python interpreter.. To end the comment you merely place another three quotes and you're back in business.

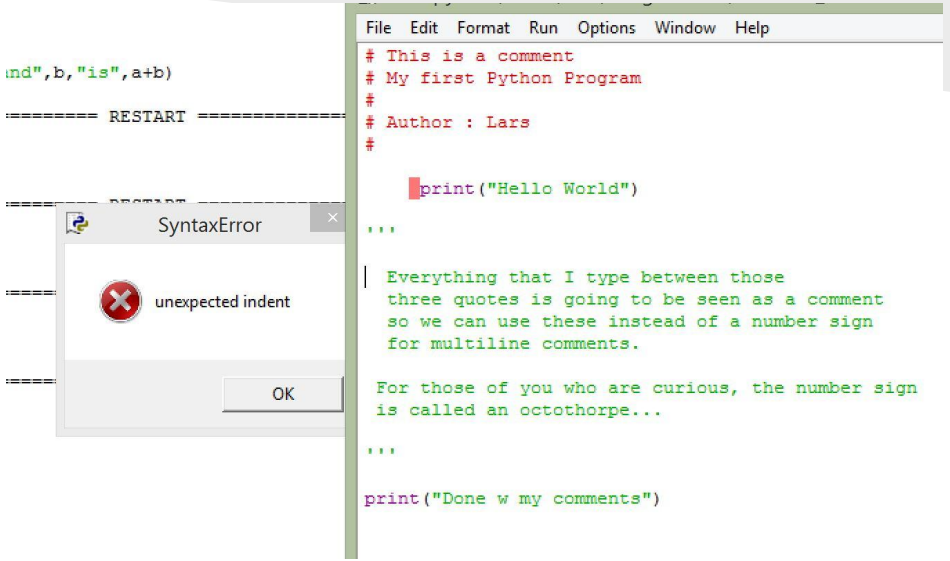
Notice that the quotes appear in the first column. Come to think about it, the commands all start there too. You'll notice that the comments inside the quotes don't though, all the space between the quotes is ignored.



Python Methodologies for Data Sciences

Whitespace

Unlike most other computer languages, Python cares about “whitespace.” When we look at code blocks we’ll see how Python uses whitespace to “delimit” lines of code that belong together. Other languages use things like braces {} to do this. In Python, to start a new block we just indent by four spaces and every line that is indented by four spaces that follows is part of the block. Don’t let this be a hangup: We’ll see more of it later, I just want you to be aware of it because it creates infuriating errors for beginning programmers who accidentally start typing in column 2.



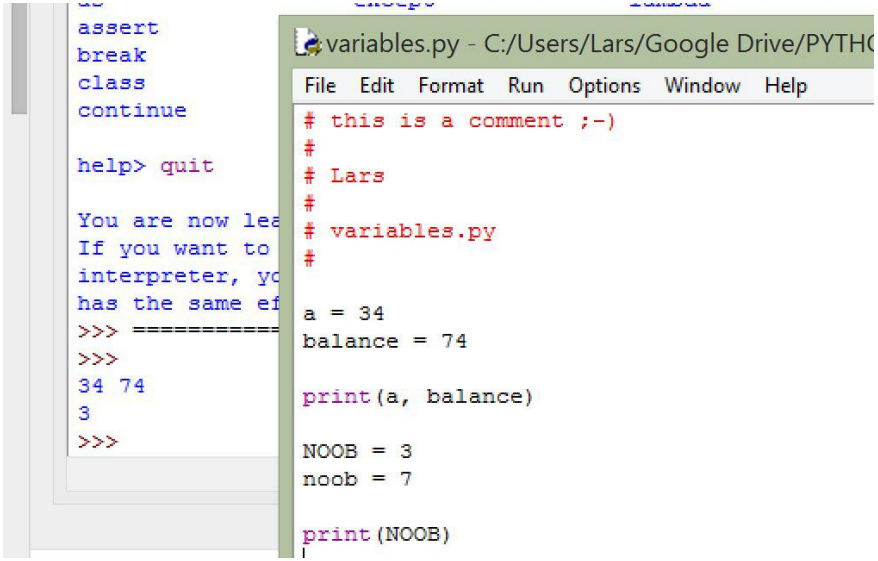
Some coders complain about Python’s note of whitespace (including the cartoon dude on the SAKAI site) but it’s silly. Programmers aren’t Jackson Pollack, it’s more important that code be readable than for programmers to be able to throw code all around the editor. Once you get going you’ll see that it’s not a hassle. Still, be aware, because it will make you nuts when it stops you from getting your program working.

Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

Okay, time to get some work done. Let's do some addition. In order to do this I will need to use a **variable**. Variables are easy, they are just data placeholders. Think of them like a named cubbyhole where you keep things. You can only keep one thing in there at a time, but you can refer to whatever is in there by the name of the cubbyhole.

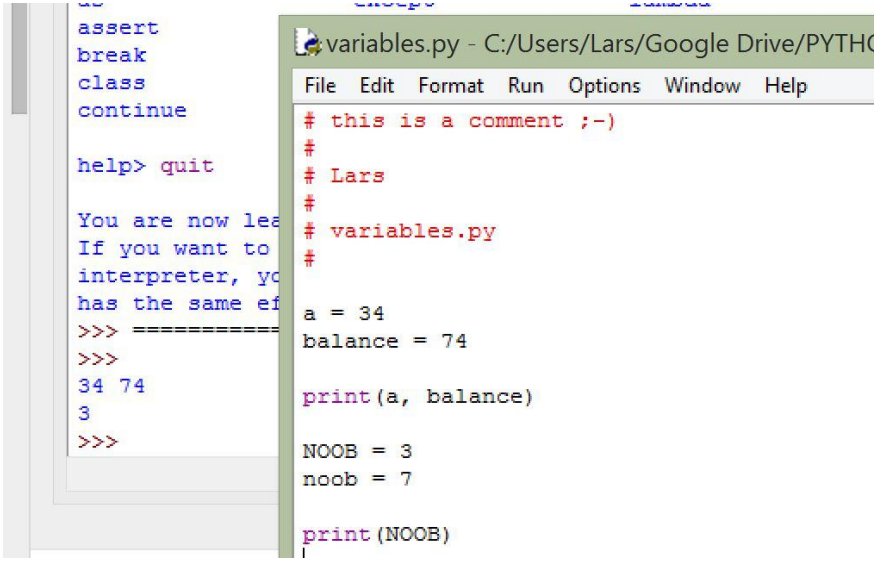
When you name variables you NEED to start with a letter or an underscore (just one underscore, not two, we'll see why later) and after that you can use letters, numbers or underscores. Avoid symbols and punctuation, it'll keep you out of trouble. Now, you SHOULD make your variable names readable (big deal when it comes to working with others) and descriptive. I can assign the value 34 to the variable "a", but it is more understandable if the variable is named "sum" or "balance".



Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

You can see that case matters as well. I assigned the value 3 to the variable NOOB and I assigned 7 to the variable noob, the same word but in lowercase. When I print, I print the uppercase version so my output shows me a 7. In Python, case matters so always keep that in mind if your variables do not print what you think they should be.



Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

Look at the code to the right carefully. There's a lot to unpack here. First, we'll see that we have two variables with descriptive names. Old Microsoft programmers used to use a system called Hungarian Notation to name their variables (and functions, we'll get to that later). In their system the first variable would be iFirstNumber and the second would be iSecondNumber.

Now we do our first assignment. We perform an arithmetic operation on our variables and place the resulting value in another variable. The first operation we perform is addition, so we call the variable sum and store the value of the addition operation there.

https://en.wikipedia.org/wiki/Hungarian_notation - **Check it out...**

variables.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/variables

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# variables.py
#

first_number = 7
second_number = 17

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

```
>>>
24 10 119 2 2.4285714285714284 49 3
>>>
```

Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

In the last line of the program we print all the values out. As you can see, sum is 24 and is correct. diff and product are also correct and pretty self explanatory. The multiplication symbol is an asterisk, it's above your 8 on your keyboard. So far so good...

Now we get to our first "Pythonic" bit of explanation. We have two kinds of division there. One operator has two slashes, not just one. A quick look at the results show that one gave only the whole number answer and the other gave the full decimal solution. We'll be explaining this in the next section, but suffice it to say for now that we have two different ways to handle whole numbers and decimals so Python provides two kinds of division.

variables.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/variables

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# variables.py
#

first_number = 7
second_number = 17

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

```
>>>
24 10 119 2 2.4285714285714284 49 3
>>>
```

Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

Two last points here. As you can see we do exponentiation with two multiplication symbols. To raise something to a power we put in two asterisks and then the power to raise the operand to, in this case 2. Seven squared is forty-nine and that's the answer printed so we're in good shape.

The last operation before the print statement may be new to you. It's called "modulo" or "modulus." It just gives the remainder of a division operation. It's incredibly useful as we will see so don't forget about it. It's just a percentage sign, the one right above the 5 on the keyboard.

https://en.wikipedia.org/wiki/Modulo_operation Check it out

variables.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/variables

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# variables.py
#

first_number = 7
second_number = 17

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

```
>>>
24 10 119 2 2.4285714285714284 49 3
>>>
```

Python Methodologies for Data Sciences

Variables - Assignment - Simple Arithmetic

One last thing. Type this in and play around with it. Just get a new file, name it variables.py and type it in real quick. Then save it and run it. Change the values of the number variables. See what happens. Make the first_number a zero. Yikes, that opens a ball of wax. Try really big numbers; I mean stupid big numbers. We're going to come back to it later so take the time to type it in and watch it run on a few different number combinations.

- I also wanted to note the "/" symbol in the print function. I put it there to extend the line and make the program look neat. All you do is type it and hit enter and IDLE will put you in the properly indented spot to keep typing. Try it.

variables.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/variables

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# variables.py
#

first_number = 7
second_number = 17

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

```
>>>
24 10 119 2 2.4285714285714284 49 3
>>>
```

Python Methodologies for Data Sciences

Data - Numbers & Words

Time to review some data basics. As we first begin to explore Python we're only going to care about four types of data. Three of them are merely numbers and words. (the fourth is Booleans, we get to them later in these slides)

In Python (and most programming languages) we call a discrete whole number like 1, 78, or -56 an Integer. My broken failing memory tells me that it's Latin for "untouched" or something and over the years has morphed into the term for whole numbers. Python will abbreviate it and refer to it as "int" in functions and when it describes the data type. Wait, data type? Keep reading...

data.py - C:/Users/Lars/Google Drive/PYTHON _ D

File Edit Format Run Options Window Help

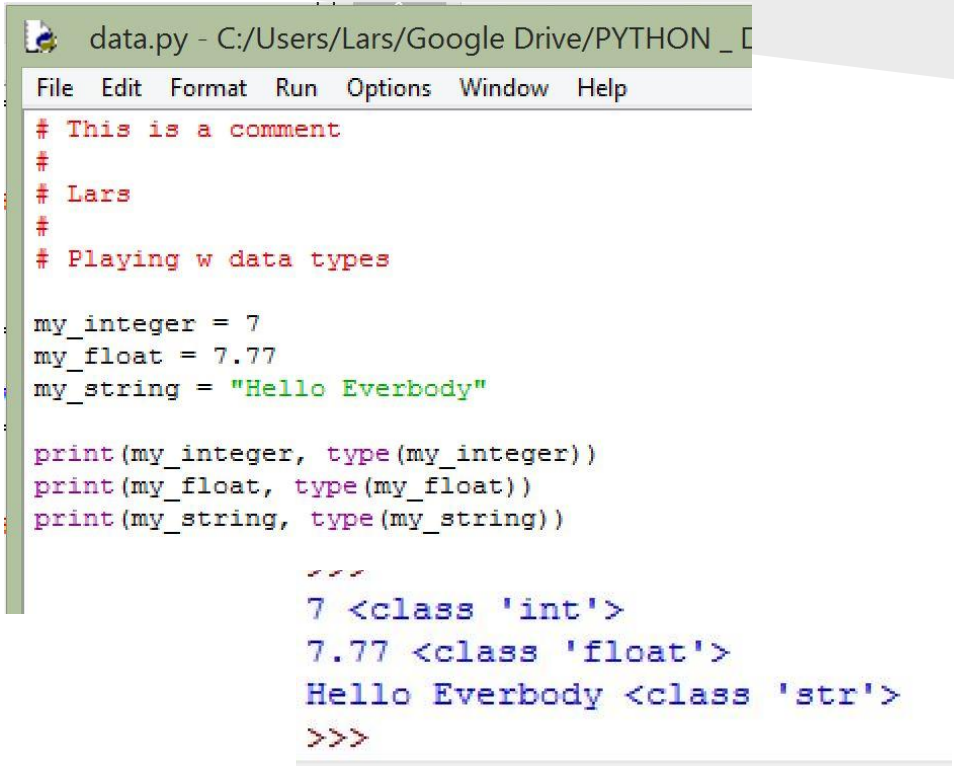
```
# This is a comment
#
# Lars
#
# Playing w data types

my_integer = 7
my_float = 7.77
my_string = "Hello Everbody"

print(my_integer, type(my_integer))
print(my_float, type(my_float))
print(my_string, type(my_string))
```

```
>>>
7 <class 'int'>
7.77 <class 'float'>
Hello Everbody <class 'str'>
>>>
```

Python Methodologies for Data Sciences

Data - Numbers & Words

A data type is just the kind of data that a variable will hold. I can give "my_integer" a 7 and Python will say "the variable my_integer now holds integers". This is called "dynamic typing". (It's also one of the things the cartoon guy makes fun of on the SAKAI site). In other languages we need to decide what we want our variables to be, forever, ahead of time and "declare" them as different types. In Python, we don't bother. This can lead to some issues down the line, but for the most part if you are a good programmer (which we all are) then dynamic typing will not be biting us in the arse.

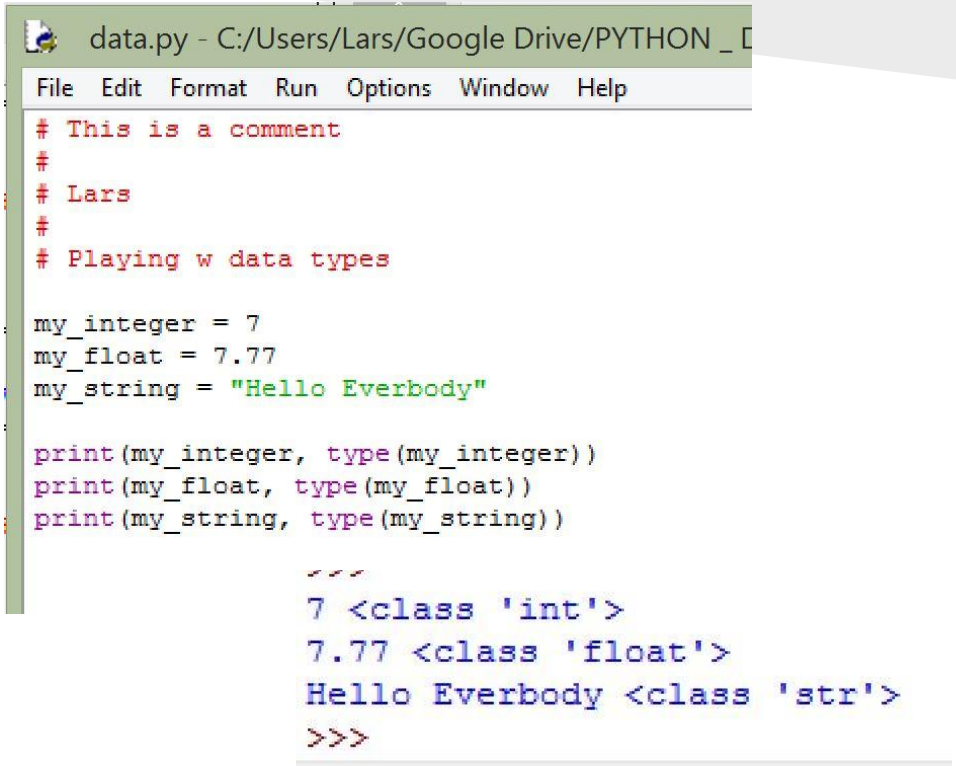


Python Methodologies for Data Sciences

Data - Numbers & Words

Now, we have two more data types to talk about. The second variable you see me declare is a floating point number or a “float”. We can go bananas and talk about significands and radix points, but for now we’ll just think of them as “the ones with decimals.” Floats are different from Integers in that they can be used for different kinds of tasks.

The natural question is “why bother?”. Why can’t we just have floats and 1 could be represented as 1.0 and we’re done with it? You know what, good point. What you need to understand it is a Computer Science history lesson about how we store numbers. Suffice it to say it’s much easier to store and work with Integers, if you can. In the dinosaur days (someday I’ll show you pictures) memory and CPU time were PRECIOUS (said like I’m Gollum).



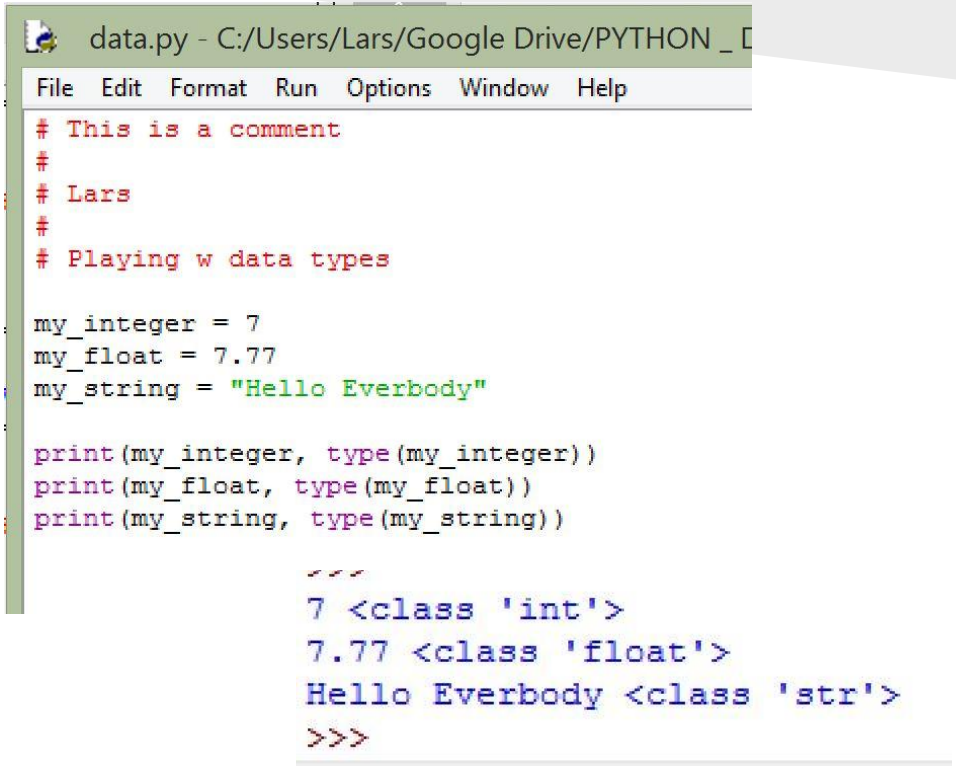
Python Methodologies for Data Sciences

Data - Numbers & Words

The third data type is strings (str). A string is just a word. We call them strings because we (computer programmers and computer science types) used to just consider single letters or characters. We called them "chars" and a word was just a "string" of chars.

We will dive deeply into strings in Unit 2, so for right now we're just going to think of these variables as the ones that hold words and not numbers.

We can also see our first utility function, the "type" function. As you can see in the print function we call the type function with our variables as input and it returns the type of the variable. In a language like Python where we have dynamic typing this will become a very useful thing to be able to do down the road.

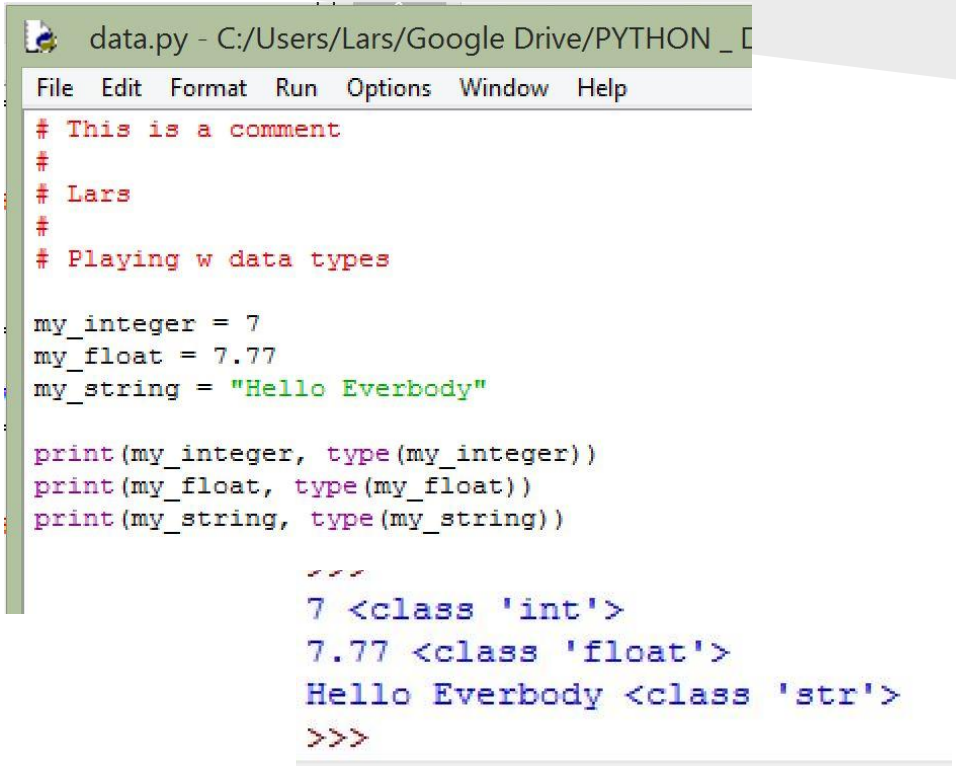


Python Methodologies for Data Sciences

Data - Numbers & Words

These simple data types are what we are going to work with as we learn Python in the early stages, but I don't want you to think that this is the limit to what we consider data. Data is anything we can use to describe a change to the state of a situation. Later, we will consider mouse clicks, mouse movement, keyboard input, colors, etc. as data. Data is all around us and it's up to the programmer to be creative and to represent it using our programming languages.

For now we care about numbers (integers and floats), words (strings) and soon we'll learn about Booleans (they're easy), but don't think this in anyway limits how we think about data. We just have to learn to walk before we run.



Python Methodologies for Data Sciences

Help and Keywords

A quick detour. If you are in IDLE type help() and hit enter. You see the Python help utility.

This system will become a valuable resource for you. I mention it here because I used it to print the keywords we're going to look at next, but I wanted you to know that the system is there for you anytime you need it.

```
>>> help()

Welcome to Python 3.4's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at http://docs.python.org/3.4/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".
```

Python Methodologies for Data Sciences

Help and Keywords

This is a list of the Python "keywords". They are special words that you cannot use when you create a variable (or a programmer defined function, we'll see that later) because they have a special meaning to the Python interpreter.

When you use them in IDLE you will see that they are displayed with a different color.

At the end of this course you will know what all thirty-three of them do.

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

Python Methodologies for Data Sciences

Help and Keywords

You don't have to memorize them or any such thing (I don't think I could name them all off the top of my head if asked), just be aware of them and know that you can go to the help system and look at them any time you need to.

I've always found it cool that an entire computer language as rich and useful as Python only has 33 keywords (it was less with "C," Python's grandfather).

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

False               def                 if                  raise
None                del                 import              return
True                elif                in                  try
and                 else                is                  while
as                  except              lambda              with
assert              finally             nonlocal            yield
break               for                 not
class               from                or
continue            global              pass
```

Python Methodologies for Data Sciences

Input from the user

Okay, time to do something next level. Remember the variable program where we played around with different arithmetic operations? If you didn't type it in and play around with it, do so now, there's a method to the madness.

Now, this is not a very useful program. You load it, run it, and it tells you a bunch of things about 7 and 17. Gets boring pretty quick, yeah? When numbers are assigned to variables explicitly inside a program we say that the numbers are "hard coded". For some variables this is fine (sometimes preferred), but for our arithmetic tests it gets tedious to keep editing the program to change numbers.

What if we could change this program so that when you ran it the program asks for the numbers to use in the operations?

variables.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/variables

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# variables.py
#

first_number = 7
second_number = 17

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

Python Methodologies for Data Sciences

Input from the user

Enter the input function. By using "input()" we can ask the user (the person running the program) for the data as we run the program.

In it, we create a prompt that the user will see and the number that is typed in will be placed in the variable it is assigned to.

As you can see below I entered the 7 and 17 as the program ran (you don't see 7 or 17 in the code anymore, right?) and we still got our arithmetic tests.

```
>>>
Enter the First number: 7
Enter the Second number: 17
24 10 119 2 2.4285714285714284 49 3
>>>
```

input.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# input.py
#
#  A program to test the input function
#

first_number = int(input("Enter the First number: "))
second_number = int(input("Enter the Second number: "))

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

Python Methodologies for Data Sciences

Input from the user

Now look closely at the first two lines after the comments. We assign our variables the results of two functions, one run inside of the other. I will explain.

If we merely said,

first_number = input("Enter the First Number")

this would be fine for printing the variable but we could not use it in arithmetic. Why? Because the results of input functions are strings. Remember data types? We can only do math on integers and floats. When we try to do math on words (strings) the code will break...

input.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# input.py
#
#  A program to test the input function
#

first_number = int(input("Enter the First number: "))
second_number = int(input("Enter the Second number: "))

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

Python Methodologies for Data Sciences

Input from the user

In this example I run the input statement alone, see? (there is no “int(“ like in the previous slide) I can print it OK (line 13), but when I try to use the variables in arithmetic operations Python responds with a big old error message (we will learn how to read these so no worries now).

What I need to do is find a way to tell Python to take the string I give it (“7” and “17”) and treat them like numbers.

```
>>>
Enter the First number: 7
Enter the Second number: 17
7 17
Traceback (most recent call last):
  File "C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py", line 16, in
 <module>
    diff = second_number - first_number
TypeError: unsupported operand type(s) for -: 'str' and 'str'
>>>
```

input.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py (3

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# input.py
#
#  A program to test the input function
#

first_number = input("Enter the First number: ")
second_number = input("Enter the Second number: ")

print(first_number, second_number)

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

Python Methodologies for Data Sciences

Input from the user

I put the “int(.....)” back now. This is how we tell Python to treat the strings that input gives it as integers. The “int” stands for integer and this is the “int” function. In programming we call this “casting”.

We can cast integers to strings (with the str function), integers to floats (with float()) and so on and so forth.

NOW we will have integers in our two number variables and be able to perform arithmetic with them.

```
>>>
Enter the First number: 7
Enter the Second number: 17
24 10 119 2 2.4285714285714284 49 3
>>>
```

input.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py

File Edit Format Run Options Window Help

```
# this is a comment ;-)
#
# Lars
#
# input.py
#
#  A program to test the input function
#

first_number = int(input("Enter the First number: "))
second_number = int(input("Enter the Second number: "))

sum = first_number + second_number
diff = second_number - first_number
product = first_number * second_number
quotient = second_number // first_number
another_kind_of_quotient = second_number / first_number

square = first_number**2
remainder = second_number % first_number

print(sum, diff, product, quotient, another_kind_of_quotient, \
      square, remainder)
```

Python Methodologies for Data Sciences

Input from the user

Now we have a useful program that can show us the arithmetic relations between any two numbers and we can choose those numbers at the time we run the program.

The four “runs” shown here show how our program handles the different inputs and also shows that even Python has no answer for a division by zero. (We will learn how to handle these things later when we discuss “error trapping” and “exception handling”).

```
>>>
Enter the First number: 7
Enter the Second number: 17
24 10 119 2 2.4285714285714284 49 3
>>>
```

```
>>>
Enter the First number: 23
Enter the Second number: 5
28 -18 115 0 0.21739130434782608 529 5
>>>
```

```
>>>
Enter the First number: 77
Enter the Second number: -23
54 -100 -1771 -1 -0.2987012987012987 5929 54
>>>
```

```
>>>
Enter the First number: 0
Enter the Second number: 3407
Traceback (most recent call last):
  File "C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/code/input.py", line 16, in
 <module>
    quotient = second_number // first_number
ZeroDivisionError: integer division or modulo by zero
>>>
```

Python Methodologies for Data Sciences

Take a Break...

Take a break. We've already done quite a bit and you have a bunch of stuff to absorb. Think about it.

You've installed Python and started using the integrated development environment IDLE to create programs. You did your "Hello World" program and went off to bigger things.

You've written programs that use comments, assigned data to properly named variables, you've performed arithmetic operations on these variables, you printed them to an output screen, you considered data types and checked the data types of your variables with the "type" function, you considered keywords, you altered your program to collect data from the user and you casted one data type (strings) into another (integers) for use in your program.

That's a lot. Congrats. Way to be. You're a programmer now. Enjoy it. Take a break and when you come back we'll finish this unit with a look at Conditionals, Boolean data types and control statements.

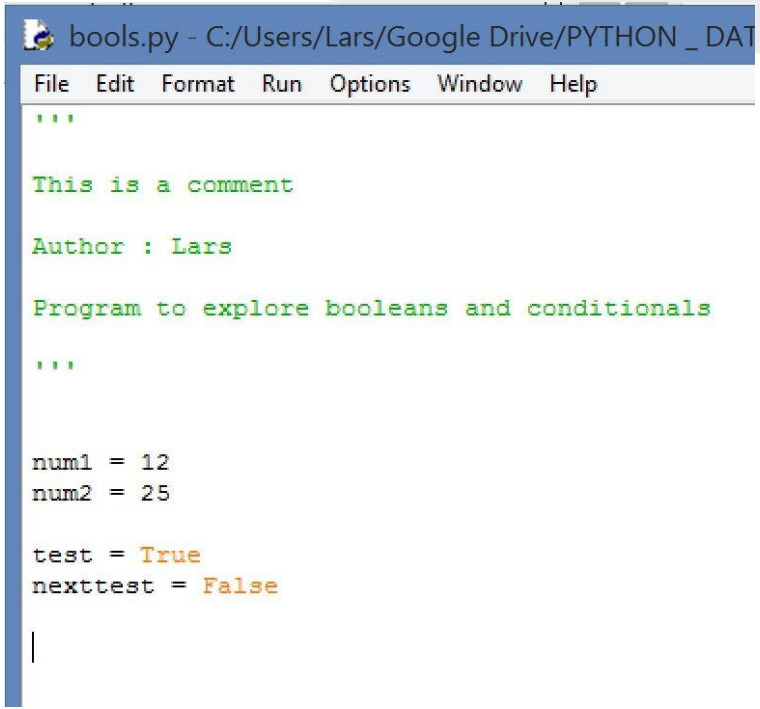
Python Methodologies for Data Sciences

Booleans and Conditionals

I hope you enjoyed your break. We only have a few more things to do for this unit. The first thing we're going to do is introduce you to the fourth data type we are going to consider in these early stages. It's called a "Boolean." Quite simply, a boolean variable holds one of two values. It is either True or False. It's that simple.

The word Boolean comes from a gentleman named George Bool. He developed a type of algebra that is based on true and false values. This algebra was the starting point for much of digital electronics and also set theory, which is near and dear to the discrete mathematical hearts of all computer scientists.

We will look at how we can use conditionals and boolean algebra operators to create True and False values and then we will learn how to have our programs perform certain tasks based on what these values are...



Python Methodologies for Data Sciences

Booleans and Conditionals

To the right you can see the six relational operators that Python uses to create conditionals.

Examine the four boolean variables I created and make sure you understand why they have the values they do. This should be fairly straight forward but it's also a new way of thinking about values and variables so take your time.

Something that often trips up new programmers is the equals conditional operator (==). It's TWO equal signs because we use a single equal sign as an assignment operator in Python (this is how it is in many other languages as well so burn it into your brain.) If you are having problems running a program later check to make sure all your equivalency conditional operators have TWO equal signs!

```
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

booltest1 = 12 > 6
print(booltest1)
booltest2 = 12 == 13
print(booltest2)
booltest3 = 12 <= 15
print(booltest3)
booltest4 = 12 != 12
print(booltest4)
```

```
>>>
True
False
True
False
>>>
```

Python Methodologies for Data Sciences

Booleans and Conditionals

We can take variables from the user and perform tests on them. As you can see here I embed the tests right in the print function. Compare this to the next slide...

```
>>>
Enter a number: 25
25 is greater than 10:  True
25 is less than 30:  True
25 is equal to 23:  False
>>> ==========================
>>>
Enter a number: 23
23 is greater than 10:  True
23 is less than 30:  True
23 is equal to 23:  True
>>>
```

```
'''
This is a comment
Author : Lars
Program to explore booleans and conditionals
'''
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

booltest1 = int(input("Enter a number: "))

print(booltest1, "is greater than 10: ", booltest1 > 10)
print(booltest1, "is less than 30: ", booltest1 < 30)
print(booltest1, "is equal to 23: ", booltest1 == 23)
```

Python Methodologies for Data Sciences

Booleans and Conditionals

Here we take the time to create the boolean variables and use them in our print statements. Either method works, but if you think you will use your value again in your program it's best to create a variable. Get a new file, name it bool.py and type in this program. When you are done run it a few times with different values.

```
>>>
Enter a number: 25
25 is greater than 10:  True
25 is less than 30:  True
25 is equal to 23:  False
>>> ============================
>>>
Enter a number: 23
23 is greater than 10:  True
23 is less than 30:  True
23 is equal to 23:  True
>>>
```

```
'''
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

booltest1 = int(input("Enter a number: "))

b1 = booltest1 > 10
b2 = booltest1 < 30
b3 = booltest1 < 30

print(booltest1, "is greater than 10: ", b1)
print(booltest1, "is less than 30: ", b2)
print(booltest1, "is equal to 23: ", b3)
```

Python Methodologies for Data Sciences

Booleans and Conditionals

We can also use conditional operators with strings! When we do the values being tested are being considered "lexicographically," which is just a complicated way to say "in alphabetical order."

Take a look at how the values and program output changed.

You may be saying, "Wait, Hello is before Howdy alphabetically, why is the first value True?"

Think of the alphabet as a number line. Now consider where "He" would be and where "Ho" would be. "Ho" is to the right of "He" so the computer thinks "greater."

```
'''
Enter a string: Howdy
Howdy is greater than Hello:  True
Howdy is less than Hello:  False
Howdy is equal to Hello:  False
>>> ================================
>>>
Enter a string: Hello
Hello is greater than Hello:  False
Hello is less than Hello:  False
Hello is equal to Hello:  True
>>>
```

```
'''
This is a comment
Author : Lars
Program to explore booleans and conditionals
with strings
'''
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

booltest1 = input("Enter a string: ")

b1 = booltest1 > "Hello"
b2 = booltest1 < "Hello"
b3 = booltest1 == "Hello"

print(booltest1, "is greater than Hello: ", b1)
print(booltest1, "is less than Hello: ", b2)
print(booltest1, "is equal to Hello: ", b3)
```

Python Methodologies for Data Sciences

Booleans and Conditionals

Before we move on to boolean algebra, let's test some of our new conditional operator knowledge..

Can you determine what the output of this program would be? If so you're likely in OK shape. If not, go back and review that last five slides and give it another go. Some people find this natural and some find it weird.

Click forward for the answers...

```
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

b1 = 4**2 > 15
b2 = -23 < -48
b3 = 23 != (12+12-1)
b4 = "Python" > "Java"
b5 = 10**4 == 100*100
b6 = 89 >= 89

print(b1, b2, b3, b4, b5, b6)
```

Python Methodologies for Data Sciences

Booleans and Conditionals

Forgive me for b4, I can't help myself sometimes.

Keep in mind that the precedence of arithmetic operators will often play a key role in many conditional expressions so keep an eye on that.

Now we will move on to Boolean algebra and start combining and choosing between boolean variables.

```
>>>
True False False True True True
>>>
```

```
# relational operators
#
#  >    - greater than
#  <    - less than
#  ==   - equal to
#  !=   - not equal to
#  >=   - greter than or equal to
#  <=   - less than or equal to
#

b1 = 4**2 > 15
b2 = -23 < -48
b3 = 23 != (12+12-1)
b4 = "Python" > "Java"
b5 = 10**4 == 100*100
b6 = 89 >= 89

print(b1, b2, b3, b4, b5, b6)
```

Python Methodologies for Data Sciences

Boolean Algebra

The final thing we have to do before jumping into control structures is to consider boolean algebra operators.

There are only three. and, or, and not. They allow us to create more complex conditionals for when we will make decisions based on these values.

For example, let's say we are game programmers and we want to print "WINNER!" to the screen when a player has more than 1000 points and is the top scorer in the game. Boolean algebra will allow us to test for both conditions in one line.

First let's look at how the three operators work.

boolalg.py - C:/Users/Lars/Google Drive/PYTHON _ DATA

File Edit Format Run Options Window Help

```
''' This is a comment
Author : Lars
Program to play with some boolean algebra
'''
# first we set our variables
v1 = True
v2 = False

# Then we construct complex boolean expressions
test1 = v1 and v2
test2 = v1 or v2

# Now we print our results
print("And:",test1)
print("Or:",test2)
```

Python Methodologies for Data Sciences

Boolean Algebra

Simply:

AND : produces a true value when both values on either side of the operator are true. If either or both of the values is false then the result of the AND operation with be false.

True and True = True
True and False = False
False and True = False
False and False = False

The only way to produce a True is if both values are True.

```
# first we set our variables
v1 = True
v2 = False
v3 = True
v4 = False

# Then we construct complex boolean expressions
test1 = v1 and v3
test2 = v1 and v2
test3 = v4 and v3
test4 = v2 and v4

# Now we print our results
print(test1)
print(test2)
print(test3)
print(test4)
```

```
...
>>>
True
False
False
False
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

Still simply:

OR : produces a true value when EITHER or BOTH values on either side of the operator are true.

True or True = True
True or False = True
False or True = True
False and False = False

The only way to produce a False is if both values are False. Notice that I only changed the ands to ors in the code example to the right.

(For you engineering types, don't worry about XOR for now...)

```
# first we set our variables
v1 = True
v2 = False
v3 = True
v4 = False

# Then we construct complex boolean expressions
test1 = v1 or v3
test2 = v1 or v2
test3 = v4 or v3
test4 = v2 or v4

# Now we print our results
print(test1)
print(test2)
print(test3)
print(test4)
```

```
>>>
True
True
True
False
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

More than simply:

NOT : produces the opposite of the value it is operating on.

Not True = False
Not False = True

It's that simple. As you can see, all of the values in the program display their opposite after the not operator is applied.

```
# first we set our variables
v1 = True
v2 = False
v3 = True
v4 = False

# Then we construct complex boolean expressions
test1 = not v1
test2 = not v2
test3 = not v3
test4 = not v4

# Now we print our results
print(test1)
print(test2)
print(test3)
print(test4)
```

```
>>>
False
True
False
True
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

We can use the “and” to create complex conditionals that lead to one value, in this case a True if the input is greater than 10 AND less than 30 (but not equal to 30, look at the 5th run...)

```
''' This is a comment
Author : Lars
Program to play with some boolean algebra
'''
# first we get a variable from the user
num = int(input("Enter a number: "))

# Then we construct complex boolean expressions
booltest1 = (num > 10 ) and (num < 30)

# Now we print our results
print("Is this number >10 and <30?: ",booltest1)
```

```
Enter a number:23
Is this number >10 and <30?: True
>>> ================================ RE
>>>
Enter a number: 24
Is this number >10 and <30?:  True
>>> ================================ RE
>>>
Enter a number: 4
Is this number >10 and <30?:  False
>>> ================================ RE
>>>
Enter a number: 39
Is this number >10 and <30?:  False
>>> ================================ RE
>>>
Enter a number: 30
Is this number >10 and <30?:  False
>>> ================================ RE
>>>
Enter a number: 12
Is this number >10 and <30?:  True
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

Change just the “and” to an “or” and see the difference. There is no way to get a False value now, the two conditions are not joined, only one of them need be true.

```
''' This is a comment
Author : Lars
Program to play with some boolean algebra
'''
# first we get a variable from the user
num = int(input("Enter a number: "))

# Then we construct complex boolean expressions
booltest1 = (num > 10 ) or (num < 30)

# Now we print our results
print("Is this number >10 and <30?: ",booltest1)
```

```
Enter a number: 23
Is this number >10 and <30?:  True
>>> ================================ R
>>>
Enter a number: 24
Is this number >10 and <30?:  True
>>> ================================ R
>>>
Enter a number: 4
Is this number >10 and <30?:  True
>>> ================================ R
>>>
Enter a number: 39
Is this number >10 and <30?:  True
>>> ================================ R
>>>
Enter a number: 30
Is this number >10 and <30?:  True
>>> ================================ R
>>>
Enter a number: 12
Is this number >10 and <30?:  True
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

Hello. Look at the booltest1 condition now. Huh?

```
''' This is a comment
Author : Lars
Program to play with some boolean algebra
'''
# first we get a variable from the user
num = int(input("Enter a number: "))

# Then we construct complex boolean expressions
booltest1 = (num > 10) or not (num < 30) and (num == 0)

# Now we print our results
print("Is this number >10 and <30?: ",booltest1)
```

```
>>>
Enter a number: 4
Is this number ????:  False
>>> ========================
>>>
Enter a number: 15
Is this number ????:  True
>>> ========================
>>>
Enter a number: 0
Is this number ????:  False
>>> ========================
>>>
Enter a number: 31
Is this number ????:  True
>>>
```

Python Methodologies for Data Sciences

Boolean Algebra

With boolean operators, as with arithmetic operators, there is an order of precedence. It's easy to remember:

NAO

Not - And - Or

But no one ever remembers it. Because of this everyone (including Zelle) recommend you put complex boolean conditionals in parenthesis so you never have to worry about such things.

If we did that here.... (that's a hint to go to the next slide)

```
''' This is a comment
Author : Lars
Program to play with some boolean algebra
'''
# first we get a variable from the user
num = int(input("Enter a number: "))

# Then we construct complex boolean expressions
booltest1 = (num > 10) or not (num < 30) and (num == 0)

# Now we print our results
print("Is this number >10 and <30?: ",booltest1)
```

Python Methodologies for Data Sciences

Boolean Algebra

```
# Then we construct complex boolean expressions
booltest1 = ((num > 10) or ((not (num < 30)) and (num == 0)))
```

Take the 4 as input and follow NAO
We evaluate the (not (num < 30)) first. That evaluates to (not True) or False.
Now we have ((num >10) or (False and (num==0))
We evaluate the "and" next.
num==0 evals to False (it's 4) so we get (False and False). This evals to False.
This leaves us with ((num > 10) or False)
Is num is greater than 10? No, it's 4, so this is False, leaving us with (False or False).
This evaluates to False. The value of the entire conditional is therefore False.

This is the less confusing way to think about it....

```
booltest2 = ((False) or ((not True) and False))
```

Python Methodologies for Data Sciences

Boolean Algebra

After those last two slide you likely think that boolean algebra is forty miles of bad road. Do not be concerned.

Part of programming with Python is learning to make code easy and readable for the next programmer who looks at it. A confusing or confuscated example like the one we just looked at would never be used in a well written Python program.

That said, sometimes it is convenient to craft complex conditionals in order to save time and compact your code.

OK. We have comparison conditionals (==, <, >, <=, >=, !=), we understand Boolean variables (True or False) and we can even create complex conditional expressions with our command of boolean algebra (Not, And, Or).

Now it's time to use this ability to start making decisions and executing code based on decision making and control structures.

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

The last thing we are going to look at for Unit one is a "control" or decision structure. That sounds complicated so forget it. They are just if-then-else structures and this is how they work.

The simplest kind, look to the right, just uses the "if." if the conditional set up after the if is true we run the block of code that' s indented four spaces and placed below the if statement. If the condition is false we do nothing. (You will notice that Python assumes the word "then." While this is classically called an if-then-else structure you never have to type the word "then" in Python.)

In the example to the right we take a number from the user. If that number is greater than 10 (not equal to) we print "This number is greater than ten!" If not, we do nothing. Note that the second print line, the one that prints "The End" is back to no indentation. It will be run no matter what, it is not part of the indented "block." Look at the next slide...

ifthen.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/

File Edit Format Run Options Window Help

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if num1 > 10:
    print("This number is greater than ten!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

You can see from the runs below that if num1 is not greater than 10 we merely skip the line that prints “This number... ten!”

This is the decision structure. You can now examine data and make decisions on whether to run code or not based on the results of your conditional statements.

```
>>>
Enter a Number: 12
This number is greater than ten!
The End
>>> ================================
>>>
Enter a Number: 7
The End
>>> ================================
>>>
Enter a Number: 10
The End
>>>
```

ifthen.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SCI/

File Edit Format Run Options Window Help

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if num1 > 10:
    print("This number is greater than ten!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

We can put parenthesis around our conditional if we desire, but we do not have to. This helps later when we look at complex conditionals

```
>>>
Enter a Number: 12
This number is greater than ten!
The End
>>> ================================
>>>
Enter a Number: 7
The End
>>> ================================
>>>
Enter a Number: 10
The End
>>>
```

*ifthen.py - C:/Users/Lars/Google Drive/PYTHON _ DATA

File Edit Format Run Options Window Help

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

Now we can look at what to do when our conditional is not true. Our control structure can have a task to perform if the conditional is true and task to perform if the conditional is not.

We use the “else” to note what to do if the condition is not true or “falls through.” Note the : after the else. This is to denote that there is a new block.

```
# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")
else:
    print("This number is 10 or less than 10!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

You can see from the run below that when the number is greater than 10 only the “greater” print line is run and when it is 10 or less the else statement kicks in and prints the “less than” print line.

```
>>>
Enter a Number: 12
This number is greater than ten!
The End
>>> ================================
>>>
Enter a Number: 7
This number is 10 or less than 10!
The End
>>>
```

```
# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")
else:
    print("This number is 10 or less than 10!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

Here we show how code is run in a block and not just a single line. When the "else" condition is true we run a "block" of code. As many lines of code that are indented like the others can be run.

```
>>>
Enter a Number: 12
This number is greater than ten!
The End
>>> ================================
>>>
Enter a Number: 7
This number is 10 or less than 10!
Yup, less than or equal to 10!
The End
>>>
```

ifthen.py - C:/Users/Lars/Google Drive/PYTHON _ DATA SC

File Edit Format Run Options Window Help

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")
else:
    print("This number is 10 or less than 10!")
    print("Yup, less than or equal to 10!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

We have a few more wrinkles to consider. The first is the "elif." It's an if within an if. We could have done it by just using another if statement, but then it would not be easy to set up our else block.

When there's a conditional with three possibilities the if-elif-else is a perfect way to handle it.

```
>>>
Enter a Number: 4
Yup, less than 10!
The End
>>> ================================
>>>
Enter a Number: 10
This number is 10!
The End
>>> ================================
>>>
Enter a Number: 17
This number is greater than ten!
The End
>>>
```

ifthen.py - C:/Users/Lars/Google Drive/PYTHON _ DAT

File Edit Format Run Options Window Help

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")
elif (num1 == 10):
    print("This number is 10!")
else:
    print("Yup, less than 10!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

Look at the second line of the > 10 block. It's an if statement under another if. This is called a "nested" if statement and is perfectly acceptable. In fact, we could put an entirely different if-elif-else structure here if we desired. Nested control structures are permissible and, frankly, very useful when you are a computer programmer. It all depends on the logic you are trying to create in your program.

If you can see what's going on here you're in pretty good shape. Get a new file, call it ifthen.py and type this in. Play with a few runs, change the nested if to another branch, play around.

```
Enter a Number: 12
This number is greater than ten!
The End
>>> ================================
>>>
Enter a Number: 7
This number is less than 10!
The End
>>> ================================
>>>
Enter a Number: 10
This number is 10!
The End
>>> ================================
>>>
Enter a Number: 25
This number is greater than ten!
It's bigger than 20 too!
The End
```

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10):
    print("This number is greater than ten!")
    if (num1 > 20):
        print("It's bigger than 20 too!")
elif (num1 == 10):
    print("This number is 10!")
else:
    print("This number is less than 10!")

print("The End")
```

Python Methodologies for Data Sciences

Conditional constructs/control statements (if-then-else)

One last thing. Remember, you can have complex conditionals for your if statements. It all depends on the logic you are trying to use in your program. If you have something to do ONLY when a value is between a range this is a good way to go.

```
Enter a Number: 14
This number is in between 10 and 20!
The End
>>> ================================
>>>
Enter a Number: 7
The End
>>> ================================
>>>
Enter a Number: 21
The End
>>>
```

```
# This is a comment
#
# Author: Lars
#
# Program to play around with if-then

# Get a number from the user
num1 = int(input("Enter a Number: "))

# If the number is greater than ten say so...
if (num1 > 10) and (num1 < 20):
    print("This number is in between 10 and 20!")

print("The End")
```

Python Methodologies for Data Sciences

A few simple exercises

(These are not your programming assignments, they are practice. Your programming assignments will be in the resource folder and are due on Spetember 17th. Do not forget them)

1. Write a program that take a number as input and prints out the square and cube of that number.

2. Using Modulo write a program that takes a number as input and prints out whether it divides evenly into 34,550.

3. Write a program that takes a number from the user and prints out whether it is negative or positive.

4. Write a program that takes a number as input. If the number is greater than 100 print its square, if less than print its cube.

5. Write a program that takes two numbers as input. Using a complex conditional print “Winner!” if both numbers are greater than 1,000.