NATIONAL GEOSPATIAL-INTELLIGENCE COLLEGE
FUNDAMENTALS OF **PROBLEM SOLVING** USING **PYTHON I**
HDN

Lesson 3: Conditional Statements

3.1. Objectives
3.2. Overview
3.3. Review
3.4. Lesson: Conditional Statements
  3.4.1. Boolean Values and Expressions
  3.4.2. If Statements
3.5. Guided Exercise: Code a Basic Calculator
3.6. Practical Exercises
  3.6.1. Practical Exercise 1: Practice Boolean Expressions
  3.6.2. Practical Exercise 2: Practice If Statements
  3.6.3. Practical Exercise 3: Guess the Output
  3.6.4. Practical Exercise 4: Odd or Even?
  3.6.5. Practical Exercise 5: Valid Question?
  3.6.6. Practical Exercise 6: Digit in Number
  3.6.7. Practical Exercise 7: Century from Year
  3.6.8. Practical Exercise 8: Date Formatting
  3.6.9. Practical Exercise 9: Valid Emails?
3.7. Appendix

3.1 Objectives

- Examine the implications of using computation to solve a problem
  - Discuss best practices for using computation to solve a problem
  - Suggest types of problems that can be solved through computation
  - Show how computation can solve a problem

- Recognize key computer science concepts
  - Identify data types used in Python scripting
  - Identify data structures used in Python scripting
  - Define variables and strings

    - Recognize how queries operate

- Demonstrate the ability to build basic scripts using Python scripting language
    - Use various data types and structures in Python scripting
    - Collect data using Python scripting
    - Extract data using Python scripting
    - Develop advanced data structures using Python scripting

---

3.2. Overview

The following material is divided into the following parts:

- Lesson
- Guided Exercise
- Practical Exercises

3.3. Review

3.3.1. Strings

Exercise: Create two string sentences.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Take the first sentence and replace all the spaces with `-` characters by using the `replace()` method.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Replace all the spaces with `/` characters by using the `split()` and `join()` methods.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Now, combine the two sentences into one string.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Find the length of the final string.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Make the final string all uppercase letters.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Access the sixth character in the string below (`':'`) using indexing, then access the last four characters (`'.com'`) using slicing.

In [ ]:

```
url = 'https://geoint.blackboard.com'
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.3.2. Numbers

Exercise: What is the difference between integers and floats?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Exercise: Can you conduct arithmetic operations between them?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Exercise: Will this this boolean expression evaluate to `True` or `False` ? Why? Try it in a code block below to check.

```
2.0 == 2
```

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Exercise: Without writing code, write what each arthimetic operator does (for the last two operators, include a description)

> Note: this exercise does not require any code, just write in regular english

In [ ]:

```
+    # operation
-    # operation
/    # operation
*    # operation
**   # operation
//   # operation
%    # operation
```

3.3.3. Casting

Exercise: Cast the following two strings to numbers and multiply them together.

In [ ]:

```
str_1 = '570'
str_2 = '76.42'
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.4. Lesson: Conditional Statements

3.4.1. Boolean Values and Expressions

References:

- Python: Boolean Values (https://docs.python.org/3.4/library/stdtypes.html#boolean-values)
- Python: Truth-Value Testing (https://docs.python.org/3.4/library/stdtypes.html#truth)
- Wikipedia: Boolean Data Type (https://en.wikipedia.org/wiki/Boolean_data_type)

A Boolean can only be one of two values: `True` or `False`. These values can be explicitly defined or defined as the result of an expression. Any Python object can be included in a Boolean expression. For example, if the variable `x` was assigned the value of `35`, then the expression `x > 40` (i.e. is `x` greater than `40`?) would evaluate to `False`.

This ability to compare and contrast one value against another is very important in computer programming. It allows computers to execute, or not execute, a task based on a condition. You can think of Booleans as your program's way of responding to yes-or-no questions. Can you think of an instance when you would need to ask your program a question, and execute different commands based on its answer?

In Jupyter Notebook code cells, you will see the capitalized words `True` and `False` always show up colored green and bold because they are reserved words in Python. Recall that each reserved word has a specific purpose in Python code and cannot be used for any other purpose.

In [ ]:

```
type(True)
```

As we mentioned above, a Boolean value can be explicitly assigned to a variable.

In [ ]:

```
my_boolean = True
my_boolean
```

What happens when you check the type of lowercase `true`? True with a small `'t'` is not a reserved word in Python because Python is case sensitive. Since `'true'` is not a reserved word or a built in function, Python interprets it as a variable or user-defined function name. And because we haven't assigned this name to anything yet in this notebook, we'll get a `NameError`.

> NOTE: Boolean values must always be capitalized.

In [ ]:

```
type(true)
```

A Boolean value can also be created with a Boolean expression. There are several types of Boolean expressions. The first type we'll learn about are comparisons.

3.4.1.1. Comparison Operators

Comparison operators are used to form a boolean expression.

Operator	Operation
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

In [ ]:

```
500 == 630
```

In [ ]:

```
bool_1 = 14 < 25
bool_1
```

In [ ]:

```
a = 2
b = 4

a < b
```

Exercise: Create your own comparison expression below. Assign the Boolean value it creates to a new variable.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Create your own comparison expression below using a different comparison operator than above.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Create a Boolean expression that tests if the weather is cold.

In [ ]:

```
temperature = 40
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Given the room capacities below, create a boolean expression that checks to see if a capacity is over 50.

In [ ]:

```
room_1 = '50'
room_2 = 65
room_3 = 10
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Why do we use two equals signs instead of a single equals sign to compare values?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

3.4.1.2. Membership Operators

Membership operators are used for a membership test. We can test for membership in any kind of sequence or collection.

Operator	Operation
`in`	Checks if the left side of expression contained in the right side of expression
`not in`	Checks if the left side of expression is not contained in the right side of expression

One type of collection is a list. We can check whether or not a value is a member of a list.

In [ ]:

```
79 in [79, 54, -9, 80, 56, 63, 67, -1, 6, 73, 9, -58]
```

In [ ]:

```
80 not in [79, 54, -9, 80, 56, 63, 67, -1, 6, 73, 9, -58]
```

A string is a collection of characters. So we can also check for membership in a string.

In [ ]:

```
'cal' in 'supercalifragilisticexpialidocious'
```

In [ ]:

```
bool_2 = 'k' not in 'supercalifragilisticexpialidocious'
bool_2
```

Exercise: Create your own membership test below.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.4.1.3. Logical Operators

Boolean expressions can also be combined to form other Boolean expressions. The `and`, `or` and `not` keywords are called logical operators and they are used to combine Boolean expressions.

Operator	Operation
`and`	Checks if two statements are *both* True
`or`	Checks if *either* of two statements are True
`not`	Negates the value of a boolean statement

The logical operators work like they do in plain language. For example, the statement "I have a black dog and I have a brown dog" is only true if I have both a black and a brown dog. However, the statement "I have a black dog or I have a brown dog" is true if I have a dog of either color.

In [ ]:

```
11 > 10 and 6 >= 10
```

In [ ]:

```
10 == 10 or 'N' in '563210S'
```

In [ ]:

```
bool_3 = not (1 == 2)
bool_3
```

Important: Note that while the logical operators work like they do in plain language, there are limitations to that. You need to include a *COMPLETE* boolean expression on either side of the operator. Below is an example of what NOT to do:

```
'N' or 'S' in '0563210E'
```

While this will not give an error, it does not result in the answer we expected. These can be the most dangerous types of bugs, because we can miss them entirely. Below is the correct way to check for multiple conditions.

In [ ]:

```
'N' in '0563210E' or 'S' in '0563210E'
```

Exercise: Combine two Boolean expressions with the `and` operator that evaluates to `True`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Combine two Boolean expressions with the `and` operator that evaluates to `False`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Combine two Boolean expressions with the `or` operator that evaluates to `True`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Combine two Boolean expressions with the `or` operator that evaluates to `False`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Use the `not` operator in order to make the last exercise's expression evaluate to `True`.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

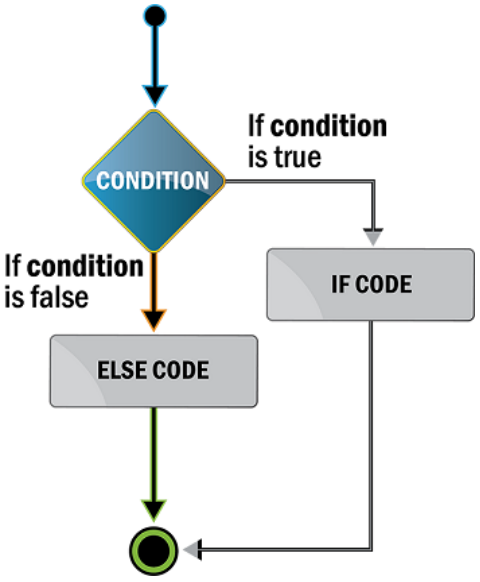
3.4.2. If Statements

References:

- Python: The if statement (https://docs.python.org/3.5/reference/compound_stmts.html#if)

- TutorialsPoint: Python IF...ELIF...ELSE Statements (https://www.tutorialspoint.com/python/python_if_else.htm)

Sometimes when we're writing code, we want to execute certain lines only under certain conditions. We hinted at this when we introduced Boolean values, saying that that's how your code responds to yes-or-no questions. Now we are going to learn how to ask yes-or-no questions in Python.



Exercise: With your neighbor, discuss some simple decisions you make everyday, and practice putting it into an if/else statement.

Thinking Questions:

> If your computer were to make this decision for you, what data would it need?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

The `if` statement is used in Python to make decisions. After the `if` keyword, we must always put a Boolean expression and then a colon (`:`). If the Boolean expression after the `if` is true, Python will execute the code indented under the `if` , then skip to the end of the `if` block. If the condition is false, Python will move on to the next statement in the block.

In [ ]:

```
temperature = 60

if temperature < 30:
    print('It is freezing! Stay inside with hot chocolate, if possible.')
elif temperature < 60:
    print('Layer up and stay warm!')
else:
    print('Have a great day')
```

Conditional execution is useful when we want to check certain values stored in our script's memory before executing certain lines of code. Below, we check to see if our `stock_value` is greater than `20000` before we execute a line that prints `SELL!` .

In [ ]:

```
stock_value = 14000
```

In [ ]:

```
if stock_value > 20000:
    print('SELL!')
elif stock_value < 8000:
    print('BUY!')
else:
    print('HOLD!')
```

Exercise: Change the value of `stock_value` above to get each print statement to run. Each time you change your variable's value, make sure you run the cell to save the new value to memory.

Exercise: In the code cell below, write your own `if` block to compare your favorite number with a classmate's favorite number. Have it print out whether your number is lower, higher, or equal to your classmate's depending on the numbers you choose.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Write an `if` block to check whether someone is old enough to vote and print a decision message.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: In plain english, write what this `if` block is doing in each line.

In [ ]:

```
my_age = 9

if my_age <= 10 and my_age >= 0:
    print ('Ticket price is free for children 10 and under')

elif my_age < 65:
    print ('Ticket price is $15')

else:
    if my_age > 125 or my_age < 0:
        print('Are you sure you input the correct age?')
    else:
        print ('Ticket price is $10')
```

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Exercise: Edit your `if` block for voter age to account for an invalid age. If they have given an invalid age, print a statement informing them of this.

HINT: As an example of an invalid age, someone's age cannot be negative or greater than 125. Additionally, age has to be a number.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Exercise: Write an `if` block using the variables `weather` and `day` that provides guidance for the day's activities based on the weather and day of week.

Example Inputs	Expected Outputs
`weather = 'cold'` `day = 'weekday'`	`'get an umbrella'`
`weather = 'cold'` `day = 'weekend'`	`'stay cozy and read'`
`weather = 'warm'` `day = 'weekend'`	`'go explore!'`

For any other inputs, your code should output `'get off work early'` .

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.5. Guided Exercise: Code a Basic Calculator

Our task is to write a script that will perform basic subtraction and addition. The inputs we get are strings of numbers and operator characters. Assume that all inputs will include two positive numbers (ints or floats) separated by either a plus (`+`) or minus (`-`) sign. Our script should output the answer to any valid input string, and the data type of the final output should be a float.

Example Inputs	Expected Outputs
'4+9'	13.0
'8-20.5'	-12.5
'100+67'	167.0

Pseudocode:

1. Determine whether the operation is plus or minus
2. Extract the two numbers (`.split()` on the operator character)
3. Perform the calculation and output the result

In [ ]:

```
input_string = '8-20.5'
```

First, let's examine our inputs. We need to be able to handle both plus and minus operations, but we don't know in advance which one of those two the input will be. This is a classic case for an `if` statement. The code below simply checks for the presence of either sign and prints a helpful message.

In [ ]:

```
if '+' in input_string:
    print('this is addition')
elif '-' in input_string:
    print('this is subtraction')
```

Now that we know which operation we're performing and we've tested our `if` block, let's begin adding the code to calculate the numbers. The first step is to break up the input string into its constituent parts. To do this, we use the string method `.split()` .

In [ ]:

```
if '+' in input_string:
    numbers = input_string.split('+')
elif '-' in input_string:
    numbers = input_string.split('-')

numbers
```

The `.split()` method gives us a list of strings. We'll cover this more in-depth in Lesson 3, but you can use

indexes to pull items out of a list in the same way you would with strings. Let's put each item in this list into its own variable.

In [ ]:

```
num_1 = numbers[0]
num_2 = numbers[1]

num_1, num_2
```

Remember that adding strings just concatenates them. That's not the behavior we want, so we'll have to cast each number to a float before we do any math with them.

In [ ]:

```
num_1 = float(num_1)
num_2 = float(num_2)

num_1, num_2
```

Now that we have two floats, we can perform the arithmetic operation. But be careful, we need to check again which operation we're supposed to be performing.

In [ ]:

```
if '+' in input_string:
    answer = num_1 + num_2
elif '-' in input_string:
    answer = num_1 - num_2

answer
```

Below, are all the steps from above combined into one cell. You can change the input and rerun the cell to see the output change.

In [ ]:

```
input_string = '8-20.5'

## Check addition or subtraction
if '+' in input_string:
    components = input_string.split('+')
elif '-' in input_string:
    components = input_string.split('-')

## Get and cast numbers
num_1 = components[0]
num_2 = components[1]

num_1 = float(num_1)
num_2 = float(num_2)

## Perform operation
if '+' in input_string:
    answer = num_1 + num_2
elif '-' in input_string:
    answer = num_1 - num_2

answer
```

3.6. Practical Exercises

3.6.1. Practical Exercise 1: Practice Boolean Expressions

References:

- Python: Boolean Values (https://docs.python.org/3.4/library/stdtypes.html#boolean-values)
- Python: Truth Value Testing (https://docs.python.org/3.4/library/stdtypes.html#truth)
- TutorialsPoint: Python - Basic Operators (https://www.tutorialspoint.com/python/python_basic_operators.htm)

Problem 1: Type out what each boolean operator does.

In [ ]:

```
>=        # definition
<=        # definition
>         # definition
<         # definition
==        # definition
!=        # definition
in        # definition
not in    # definition
and       # definition
or        # definition
```

Problem 2: Create two different boolean expressions that evaluate to True using different comparison operators.

In [ ]:

```
### YOUR CODE GOES HERE ###
```

Problem 3: Create a different boolean expression that evaluates to True using a membership operator.

In [ ]:

```
### YOUR CODE GOES HERE ###
```

Problem 4: Create a different boolean expression that evaluates to False using a membership operator.

In [ ]:

```
### YOUR CODE GOES HERE ###
```

Problem 5: Create two unique boolean expressions using logical operators and any other operators.

In [ ]:

```
### YOUR CODE GOES HERE ###
```

3.6.2. Practical Exercise 2: Practice If Statements

References:

- Python: if Statements (https://docs.python.org/3.4/tutorial/controlflow.html#if-statements)

Problem 1: When do you use `if` statements?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Problem 2: What does the `elif` keyword allow you to do and how many times can you use it?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Problem 3: Do you need to use an `else` statement when you build an `if` block? If you use an `else`, can you use a boolean statement with it? Why or why not?

In [ ]:

```
## YOUR ANSWER GOES HERE -- NO CODE NECESSARY ##
```

Problem 4: Build an `if` block that checks your checking account balance. If the balance is under `500`, print an alert message. If it is over `5000`, print a recommendation to invest some funds or move some funds to savings.

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.3. Practical Exercise 3: Guess the Output

References:

- Python: if Statements (https://docs.python.org/3.4/tutorial/controlflow.html#if-statements)

Without running the code below, what is the final value of `number`?

```
number = 10

if number == '10':
    number = number + 9
elif number > 1 and type(number) == float:
    number += 5
elif number % 2 == 0:
    number = number ** 2
else:
    number *= 2.5

print(number)
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Follow-Up Questions:

1. How does each of the Boolean expressions in the if block evaluate (`True` or `False`)?
2. What is the data type of the answer?

3.6.4. Practical Exercise 4: Odd or Even?

References:

- Python: Boolean Values (https://docs.python.org/3.4/library/stdtypes.html#boolean-values)
- Python: Truth Value Testing (https://docs.python.org/3.4/library/stdtypes.html#truth)
- Python: if Statements (https://docs.python.org/3.4/tutorial/controlflow.html#if-statements)
- TutorialsPoint: Python - Basic Operators (https://www.tutorialspoint.com/python/python_basic_operators.htm)

Write a Python script that will take an integer and print whether it is even or odd.

Example Input	Expected Output
2	Even
86	Even
107	Odd
9	Odd

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.5. Practical Exercise: Valid Question?

Write a Python script that will take a string and check if it is a valid question.

HINT: A valid question will end with a question mark.

Example Input	Expected Output
This is a sentence.	False
This is an exclamation!	False

Example Input	Expected Output
`Is this a question?`	`True`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.6. [Challenge] Practical Exercise 6: Digit in Number

Write a script that, given a number (int or float) and a digit (int), will output `True` if the number contains the digit and `False` otherwise.

Example Input	Expected Output
`number = 2345.6` `digit = 4`	`True`
`number = 10000` `digit = 3`	`False`
`number = 3.14159` `digit = 8`	`False`
`number = 3.14159` `digit = 9`	`True`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.7. [Challenge] Practical Exercise 7: Century from Year

Write a script that, given a year (int), outputs the century as defined in the table below.

Year Range	Century
1-100	1
101-200	2
...	...
1901-2000	20
2001-2100	21

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.8. [Challenge] Practical Exercise 8: Date Formatting

Problem 1: Write a script that, given a month (str) and a date (int), will output a string that combines both date and month like so: `the 22 of June` .

Example Input	Expected Output
`month = 'June'` `date = 22`	`the 22 of June`
`month = 'October'` `date = 5`	`the 5 of October`
`month = 'March'` `date = 18`	`the 18 of March`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Problem 2: Extend your script so that the output string contains the appropriate suffix for the date (`'st'` , `'nd'` , `'rd'` , `'th'`).

Example Input	Expected Output
`month = 'June'` `date = 22`	`the 22nd of June`
`month = 'October'` `date = 5`	`the 5th of October`
`month = 'March'` `date = 1`	`the 1st of March`
`month = 'August'` `date = 3`	`the 3rd of August`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

Problem 3: Extend your script to output a helpful error message if the date is not between 1 and 31 (inclusive).

Bonus: Check for valid date values based on the month (e.g. February dates can only be between 1 and 29).

Example Input	Expected Output
`month = 'June'` `date = 32`	`Invalid date: the date must be between 1 and 31`
`month = 'October'` `date = -8`	`Invalid date: the date must be between 1 and 31`
`month = 'March'` `date = 1`	`the 1st of March`

Example Input	Expected Output
`month = 'August'` `date = 3`	`the 3rd of August`

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.6.9. [Challenge] Practical Exercise 9: Valid Emails?

Given an email address, check that it is valid. For the purposes of this exercise, an email address is valid if it has one `@` character and ends with either `.com`, `.edu`, `.gov`, `.eu`, `.uk`, or `.in`. HINT: You have already been given a list of all the valid endings. Remember, that you can conduct membership testing against lists. HINT: Try using `.split('.')` on the email address. You can access elements of a list by their index just like you access characters in strings.

Example Input	Expected Output
`email = 'president@whitehouse.gov'`	`Valid Email!`
`email = 'jeff@amazon.slk'`	`Invalid email`
`email = 'mark$facebook.com'`	`Invalid email`

In [ ]:

```
valid = ['com','edu','gov','eu','uk','in']
```

In [ ]:

```
## YOUR CODE GOES HERE ##
```

3.7. Appendix

Boolean Operators

Operator	Operation
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than

Operator	Operation
`<=`	less than or equal to
`in`	Checks if the left side of expression contained in the right side of expression
`not in`	Checks if the left side of expression is not contained in the right side of expression
`and`	Checks if two statements are *both* True
`or`	Checks if *either* of two statements are True
`not`	Negates the value of a boolean statement

Assignment Operators

Operator	Operation
`+=`	adds right operand to the left operand, then assigns the result to left operand
`-=`	subtracts right operand from the left operand, then assigns the result to left operand
`*=`	multiply right operand by the left operand, then assigns the result to left operand