**Deadline: 9 pm**

* **Bits:** A bit or a binary digit, is the smallest unit of data that a computer can process and store. A bit can be one of 2 physical states, such as either 0 or 1, yes or no, and true or false.
* **Bytes:** A byte is a unit of data that is eight bits long. It is used by computers to represent a character such as a letter or number symbol.
* **Hexadecimal:** Hexadecimal is a numbering system with base 16. There are 16 symbols or possible digit values from 0 to 9, followed by six alphabetic characters -- A, B, C, D, E and F. It can be used to represent large numbers with fewer digits.

**Nibbles**

Binary Numbers:

1. **Unsigned Integer:** Numbers without + or - sign, only representing the magnitude not the direction. Used when we know that the value we are storing will always be non-negative
   1. Ex. 1, 2, 3, 4
2. **Signed Integer:** The default where the variable can hold positive or negative numbers
   1. Ex. -1, 6, -29, 1000
3. **Floating Point:** Positive or negative whole number with a decimal point. Used to represent real numbers and is written with a decimal point dividing the integer and fractional parts
   1. Ex. 1.23, 87.425, and -9039454.2, NOT 101, -3, 18203

Binary Data Abstractions:

1. **Boolean:** Variable type that represents one of two values: True or False.

A = 200

B = 33

If b > a:

print(“b is greater than a”)

If a > b:

print(“a is greater than b”)

1. **ASCII:** Stands for American Standard Code for Information Interchange. ASCII was first created when all people wanted computers to be able to show was numbers, letters, punctuation, and non-printing commands(Enter, Delete, F1).. All ASCII encoded character can be represented by 1 byte, or 8 bits for a total of 2^8 = 127 different total characters.
   1. Ex. lowercase “h” character → 104 (D) → 01101000(B)
2. **Unicode:** today, people use emojis and new symbols like other languages and stuff and computers needed to be able to represent all that. Unicode is a new standard defining an association between characters and even more numbers. Each unicode character (utf8) is 4 bytes, or 32 bits, enough for 2^32 different characters. Most modern programming languages represent strings as utf8 encoded characters (hex) because people use these new symbols in their code, but old languages like C still associate either character type with a single byte integer
   1. Ex. "Hello" → U+0048 U+0065 U+006C U+006C U+006F
3. **RGB:** RGB (red, green, and blue) refers to a system for representing the colors to be used on a computer display.These 3 colors can be combined in various proportions to obtain any color in the visible spectrum.
   1. Ex. rgb(255, 0, 0) = red because red is set to its highest value (255), and the other two (green and blue) are set to 0.

Data Compression

1. **Lossy**: a technique that reduces file size by discarding less important information
2. **Lossless** (unit 3): Every bit of data originally in a file remains after it is uncompressed, and all the information is restored.

* Variables:

An abstraction inside a program that holds a specific value or meaning defined by the programmer.

* Data Types:

Integer- highScore (involves math, save as integer or numbers)

String- firstName (name is text, so it is a string)

Boolean- isSunny (2 options, true or false)

String- phoneNumber (no math, just numbers)

* Arithmetic Operators

Plus indicated addition (a + b)

Minus indicates subtractions (a - b)

Asterisk indicated multiplication (a \* b)

Slash indicated division (a / b)

* Managing Complexity with Variables
  + Lists: Allows you to complete a process for each value in the list, or store multiple values to one variable.
  + 2D Lists: Array within an array. You can pull single arrays through using index and variable commands.
    - Ex: T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]
* Dictionaries: Allows the storage or data keys and values (assigning a certain piece of data to fit the category it is assigned)
* Class: A particular data structure.
* Algorithms: An algorithm is a finite set of instructions that accomplish a specific task, us as humans, do algorithms on a daily basis.
* Sequence: A specific order a process is completed, which impacts the output.
* Selection: Programmer decides between 2 different functions
* Iteration: Repetition of a process
  + For loop: repeats a section of code a set number of times
  + While loop: repearts a section of code an unknown number of times, until the code is told to break
* Expressions: A piece of syntax in coding

Leonard’s Section:

* Comparison Operators
  + Operators used in programming languages which compare two different values. These can often be used to set up statements which return a boolean value (true/false), which can be stored in variables. All examples below are in [Python](https://docs.python.org/3/reference/expressions.html) and were checked using the bool() command, which returns the True/False output.

| Operator | Function | True output example | False output example | Notes |
| --- | --- | --- | --- | --- |
| == | Checks if two values are equal to each other | “Hrar” == “Hrar”  1 == 1.0 | “Hrar” == “hrar”  1 == 2 | If strings are compared, this is case-sensitive. Follows numerical equivalence if a float and integer is used. Cannot use leading zeros for integers/floats. |
| != | Checks if two values are not equal to each other | “Hrar” != “hrar”  1 != 2 | “Hrar” != “Hrar”  1 != 1.0 | Similar to above. |
| < | Checks if the left value is less than the right value | 1 < 2  “zz” < “hi” | 2 < 1  “az” < “hi”  2 < 2 | If strings are compared, the string appearing last in alphabetical order is considered less. **float(‘NaN’)** and **decimal.Decimal(‘NaN’)** will yield false regardless of what they are compared to. |
| > | Checks if the left value is greater than the right value | 2 > 1  “az” > “hi” | 1 > 2  “zz” > “hi”  2 > 2 | Similar to above |
| <= | Checks if the left value is less than or equal to the right value | 1 <= 2  “zz” <= “hi”  2 <= 2 | 2 <= 1  “az” <= “hi” | Similar to above |
| >= | Checks if the left value is greater than or equal to the right value | 2 >= 1  “az” >= “hi”  2 >= 2 | 1 >= 2  “zz” >= “hi” | Similar to above |
| in/not in | Checks if the specified value is an element in the specified list. **in** returns true if the element is in the list; **not in** returns true if the element is not in the list.  Example: List = [1, 3, 5, 7, 9, “hi”] | 1 in List  3 in List  5 in List  “hi” in List  “What” not in List  “Why” not in List | 2 in List  4 in List  6 in List  9 not in List  7 not in List  “Weird” in List | As usual, strings are case-sensitive. May be useful for conditional statements. |

* Booleans Expressions and Selection
  + The value of a boolean variable (True/False) can be used as conditions in selection (often known as if statements). The condition of the if statement is the boolean variable; if its value is True, the condition passes. A possible function of this is in Python code below

# Assume the grade was calculated by a computer program

# The bool command is here to make a boolean variable

# In a larger computer program, the value was likely

# generated by previous commands

Grade = 89.49

isA = bool(Grade >= 90)

if isA:

print(“Yay!”)

else:

print(“hmm”)

OUTPUT: hmm

* Booleans Expressions and Iteration
  + Certain forms of iteration (specifically while loops in Python) can use a boolean variable as a condition (similar to selection commands). As you may expect, the while loop executes commands until the value of the variable is false. In terms of a computer program, this could perhaps be used to execute commands critical for maintaining a certain process until that process is no longer needed. This could also potentially be used to periodically send notifications until a certain condition is met (ex: an assignment is submitted)
  + Here’s some informal python code:

num = 10

correct = bool(num > 0)

while correct:

print(num)

num = num - 1

correct = bool(num > 0)

OUTPUT:

10

9

8

7

6

5

4

3

2

1

* Truth Tables: A table for a logical operator (ex: AND, OR, XOR) containing each variable and all possible input and output results of that operator.

AND TABLE: All inputs must be true to return output of true.

| Input 1 | Input 2 | Output |
| --- | --- | --- |
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

OR TABLE: At least one input must be true to return output of true.

| Input 1 | Input 2 | Output |
| --- | --- | --- |
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

XOR TABLE: Exactly one input must be true to return output of true.

| Input 1 | Input 2 | Output |
| --- | --- | --- |
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | False |

* Characters
  + The units which make up a string. These can be letters, numbers, special symbols (!@#$%^&\*), or even spaces. Regardless, if one types, each keystroke outputs one character.
  + Ex: “keyboard hi!!!” has 14 characters: 10 letters, three exclamation marks, and a space.
* Strings
  + A variable data type which consists of a combination of letters, numbers, and other special characters (ex: @#$%^&\*). They are seen inside quotation marks, and operators cannot affect any numbers in the string.
  + Ex: stringVar = “Hello World!”
  + “Hello World!” is a string.
* Length
  + The number of elements in a list. Here’s an example in Python:
  + numList = [1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1]
  + This list has a length of 17 because it has 17 elements. Each element is separated by a comma.
* Concatenation: the act of joining two strings into a single string. Often, these strings are printed to an output terminal/console. Here’s an example in pseudocode:
  + concat(“tri”, “llion”)
  + This concatenates the strings “tri” and “llion”, making a new string “trillion”.
* Upper, Lower, Traversing Strings
  + The first two (upper and lower) seem to be referring to Python methods. They are quite simple. I think it would be easiest to explain with a demonstration:

String = “WhYaReWeDoInGtHiS”

print(string)

print(string.upper())

print(string.lower())

Output:

WhYaReWeDoInGtHiS

WHYAREWEDOINGTHIS

whyarewedoingthis

Relatively intuitive. The upper() method changes all lowercase letters in a string to uppercase, while the lower() method does the opposite. A potential reason for this would be ensuring uniform case (especially of uppercase when making important headings like page/screen titles)

* Traversing strings behaves highly similar to iterating elements in a list, except that instead of iterating through a list, it’s a string. Elements in a list are now the characters of a string. A potential application of this could be to search all strings for a specific substring. Here’s some example python code:

#Very informative example here: #<https://www.geeksforgeeks.org/iterate-over-characters-of-a-string-in-python/>

String = “Thirty Trillion Dollars”

for i in String:

if i == “T”

print(“T detected!”)

Output:

T detected!  
T detected!

* Python If, Elif, Else conditionals;
  + Python if statements execute associated lines of code if the **if condition** is satisfied.
  + If the **if condition** is not satisfied, an Elif statement is executed as fallback… with another condition and associated commands
  + If all associated **Elif conditions** are not satisfied, there can be an else statement which unconditionally executes associated commands.

Ex: input = input(“enter a string”)

if input == “Hi”:

print(“Hello World!”)

print(“How are you doing today?”)

elif input == “Why”:

print(“That’s a good question to ask about everything that happens.”)

elif input == “Who are you”:

print(“I’m an anonymous guest. Pleased to meet you.”)

else:

print(“Have a great day!”)

* This code uses if statements and elif statements to return strings for three special inputs (“Hi,” “Why,” “Who are you”). If the input were any of those strings, the associated command would be executed. Otherwise, the command associated with the else statement (which prints “Have a great day!”) would be executed.
* Nested Selection Statements: When a program can choose from many paths to lead to a result, think of flowcharts where one condition must be satisfied before another can be picked
* Python For/While loops with Range with List: There are two ways loops can be used one is with a range and the other is with a list. When using the range method a variable is iterated a set number of times based on the range (typically an integer value), while with the list method the loop is iterated based on the length of a list.

Range:

i = 0

while i < 5:

print(i + 1)

i += 1

Output: 1, 2, 3, 4, 5

List:

list = [“a”, “b”, “c”]

i = 0

while i<len(list):

print(list[i])

i = i + 1

Output: a, b, c

* Combining loops with conditionals to Break: Makes it so that the loop will stop as soon as a condition is met
* Continue: A statement used in while and for loops which starts the code from the beginning, in the image below you can see how the “h” in “Python” was not printed since the continue statement skipped the print(letter) and made the code start from the beginning

for letter in “python”:

if letter == “n”:

continue

print(letter)

Output: pytho

* Procedural Abstraction: A model of what the code should do but it doen’t tell how to do it. It’s basically using a procedure to name the idea.
* Python Def: It defines a function and makes code simpler by allowing for the code in the function to be called
* Procedures: Code that doesn’t return a value but allows for a specific task to be completed
* Parameters: Values that are passed into functions when they are defined in addition(a + b), a and b will be given a value and that value is a parameter
* Return Values: Values that are outputed when a function is completed. An example of this was when we were using console.log(“message”) when testing in AppLab