Fundamental Computing – Practical Content Review A

- **Primitive Data Types**
  - Integer
  - Float
  - String
  - Boolean

- **Basic Operations on Primitive Data Types**
  - Arithmetic operators

+	–	*	/	%	//	**

  - Logical operators

==	!=	<	<=	>	>=	or	and	not	in

  - String operators

+	*

  - Precedence
    - Use brackets (i.e., "(" and ")") when unsure
    - Review: https://docs.python.org/3/reference/expressions.html#operator-precedence

- **Variables and Assignment**
  - Legal variable names
    - Cannot begin with a digit
    - Cannot include operator symbols
    - Cannot be reserved words (e.g., or, and, not, in, is, def, return, pass, break, continue)
    - Should not be built-in function names (e.g., print, input, range, len, min, max, int, str)
  - Assignment
    - In the form: <variable name> = <expression>
      - <expression>: <operand> <operator> <operand> | <expression> <operator> <expression>
      - <operand>: <literal value> | <variable> | <function call>
      - Examples of literal values: -1, 0, 1, 3.14, "hello", True, None, etc.
    - Updates the lookup table such that the variable in question now exists, and is associated with a value at the specified memory location

- **Input and Output**
  - Printing to the terminal with the print(...) statement
    - Should always be used as a function call
    - Should always only take in a single string corresponding to the desired output
  - Requesting user input via the input(…) statement
    - Should always be used as part of an assignment statement
    - Should always only take in a single string corresponding to the desired output (i.e., instruction to the user)

- **Type Casting**
  - Converting a value of one type to another (e.g., string to integer); performed by calling:
    - int(…)
    - float(…)
    - str(…)
    - bool(…)

- **Control Structures**
  - if-elif-else statement
    - First conditional execution should be in the form:
      - if <Boolean Value>:
        <CODE BLOCK>
    - Intermediate conditional executions should be in the form:
      - elif <Boolean Value>:
        <CODE BLOCK>
    - Final block should be in the form:
      - else:
        <CODE BLOCK>
    - Note that all blocks in an if-elif-else statement are mutually exclusive (i.e., only 1 of them will be executed)
  - while loop
    - Should be in the form:
      - while <Boolean Value>
        <CODE BLOCK>
  - for loop
    - Should be in the form:
      - for <variable> in <collection>
        <CODE BLOCK>
      - Note that in the case of a for loop, the specified variable will take on each value in <collection> (following the order of the collection) – i.e., for the i-th iteration of the loop, the variable will take on the i-th value in the collection
      - The values of collection that are iterated are assigned prior to the first iteration, and are NOT subject to changes in the specified collection – i.e., even if the collection changes throughout the various iterations, this will not affect the values that the specified variable with iterate through
  - Note that the flow within loops may change by using:
    - break
      - Stops the loop entirely when called; code after the loop continues
    - continue
      - Stops the current iteration of the loop when called; code in the next iteration continues
  - Helpful built-in functions to assist with control structures:
    - range(…)
      - range(a) is a collection (0, 1, 2, …, a)
      - range(a, b) is a collection (a, a+1, a+2, …, b-1)
      - range(a, b, c) is a collection (a, a+c, a+2c, a+3c, …, b-1)
    - len(…)
      - len(L) returns the number of elements in the collection L

- **Strings and Lists**
  - These are both collections
  - Strings are immutable (value cannot be modified)
  - Lists are mutable (value can be modified)
  - Indexing
    - L[i] returns the i-th element of a string or list (starting at index 0)
  - Reverse indexing:
    - L[-1] refereces the last element in L
    - L[-2] references the second last element in L
    - L[-a] references the a-th last element in L (i.e., a elements from the back)
  - Slicing
    - L[a:b] returns the sub-collection (L[a], L[a+1], L[a+2], …, L[b-1])
    - L[a:b:c] returns the sub-collection (L[a], L[a+c], L[a+2c], L[a+3c], …, L[b-1])
    - Shortcuts:
      - L[:] returns all elements in L
      - L[:a] returns all elements in L up to (but not including) index a
      - L[a:] returns all elements from index a till the last element in L
  - Copying lists
    - Since lists are mutable, they are copied by reference; to copy a list, you need to manually copy each element into the new list, or use a slice of all elements
  - List comprehension
    - For example: [x for x in range(100)]
    - Note that there are many complex usages for list comprehension; you should explore these carefully
  - You may casting from list to string and vice versa; be sure to explore how these work
  - Helpful string methods:

index(…)	lower()	upper()	replace(…)	join(…)	format(…)	isspace()
isalnum()	isalpha()	isdigit()	isdecimal()	isnumeric()	islower()	isupper()

    - You should review all the above functions and learn how to use them
  - Helpful list methods:

append(…)	pop(…)	index(…)	reverse()	sort()

    - You should review all the above functions and learn how to use them

- **Dictionaries**
  - A special form of list that allows each element index to be a specified value; for example D = {“a”: 1, “b”: 2}, where D[“a”] will return 1 and D[“b”] will return 2
  - Each element of the dictionary thus has 2 parts:
    - Key – the index value to reference if the correspond value is desired
    - Value – the value stored
    - From the example above, “a” and “b” are keys, while 1 and 2 are their corresponding values
  - Helpful Dictionary Methods:

keys(…)	values(…)

    - You should review all the above functions and learn how to use them

- **Functions**
  - Utilised to write a specific module or unit of code
  - Defined by using def
    - For example:

      def <function name>(<comma-separated parameter list>):

        <CODE BLOCK>
    - A return line will cause the function to stop executing and return the specified value
      - For example:

        return True
    - When no return statement is specified or if a function resolves without a return value, a None value is automatically returned
      - Note that the None value is a special value signifying "no value"
  - Execution of functions utilises what is known as a call stack
    - The most important aspect of the call stack is that each allows each function call to have its own lookup table
    - Thus, unless a variable is specified as global, a function should only utilise local variables (i.e., variables either defined as parameters of the function or within the function itself)

- **Recursion**
  - Instead of using a loop to repeat functionality, uses nested function calls
  - Requires the specification of:
    - Base case(s)
    - Recursive case(s)
  - Wrapper-based recursion may be used when some functionality need only be run once; i.e, this corresponds to the usage of a main calling function that calls the actual recursive function

- **Other Helpful Built-in Functions**

abs(...)	bin(...)	chr(...)	list(...)	hex(...)	map(...)	min(...)	max(...)
oct(...)	ord(...)	pow(...)	round(...)	reversed(...)	sorted(...)	sum(...)	tuple(...)

  - You should review all the above functions and learn how to use them

- **Helpful Modules**

random	math	datetime	re	csv

  - You should review all the above modules and learn how to use them