IDX G9 Programming Essentials H
Study Guide Issue 2
By Loren and Forrest, Edited by Cathy

Contents:

Intro to Python

Software

- Codes of instructions for computer hardware operations
- Created through programming
- Acts as the interface between human and hardware, where it translates human input into the form that hardware can understand and vice versa
    - Hardware can only understand binary system

Steps to Solve a Programming Problem

- Defining the problem
- Planning the solution
- Writing the code

- Testing and **debugging** the program
    - **Debugging**: removing mistakes, also known as “bugs” in coding

Programming Languages

- Python, Java, C++
    - Python is easy to code and has plenty of libraries
- Python3 is the version of Python used throughout the course
- 2 modes: **interactive** and **script mode**
    - **Interactive mode**: results are returned after clicking “enter”
        - Code has to be complete (parentheses have to be closed, etc.)
        - After clicking “enter”, code cannot be edited unless a new line is started
            - A new line is marked the symbols >>>
    - **Script mode:** multiple lines of code can be enacted at the same time
        - Note that you need to create a file!
            - Click “File” in the upper-left corner (or press Ctrl + N) → “New file”
            - File needs to be saved (Ctrl + S) and run (F5/“run” button)
            - Code can be edited without needing to open new files
        - Lines will not be marked by >>>

Variables

- Containers for storing **data values**
    - **Data values** can be numerical or textual
- Used for readability and to avoid repeating long values
    - Helps in the **debugging** process
    - Shorter codes also make the program run faster
- A new variable is created the moment it is **assigned** a value
    - **Assignment** refers to the declaration that the variable is a certain value

Naming Rules

- A variable name can only contain alphabets, numbers, and underscores
    - Cannot start with a number (can start with underscores or letters)
    - Cannot contain special symbols, such as @, #, $, etc. or spaces
    - Cannot start with **reserved keywords**

        - **Reserved keywords**: built-in commands that already have a purpose, such as True, False, if, finally, etc.
        - To see a full list of reserved keywords, type “import keyword”, return to a different line, then type “keyword.kwlist”
    - Case sensitive (Hello is a different variable from hEllo)
    - Should be short but descriptive
- There are 3 major naming conventions
    - **Camel Case**: first letter of every word starting from the 2nd word is in uppercase
        - Makes compound names more readable
        - E.g. numberOfDonuts, camelCase
    - **Snake Case**: each word is lowercase and separated by an underscore (_)
        - E.g. number_of_donuts, snake_case
    - **Pascal Case**: first letter of every word (starting from the 1st word!) is in uppercase
        - E.g. NumberOfDonuts, PascalCase

Assigning Values

- Values are **assigned** with the equal (=) operator
- **Multiple assignment**: the assignment of a value or multiple values to multiple variables
    - E.g. a = b = c = 10 will assign the value 10 to a, b, and c
    - E.g. a, b, c = 1, 2, 3 will assign each variable to the corresponding values
        - Equivalent to a = 1, b =2, and c = 3 in separate lines

Data Types

- **Integers**: a number without decimals (the same definition as with math)
    - E.g. 10, 20, 918
- **Floating-Point Number (Float)**: a number with decimals
    - E.g. 1.0, 2.9, 18.102
    - Approximations of real numbers
        - 2/3 returns 0.6666666666666666, not 0.666…6 or other variations
- **String**: a textual value indicated with quotation marks (either “” or ‘’)
    - E.g. “hello world”, ‘10’, “10.9” (note the quotation marks around the numbers, making it a string, not an integer or float)

- **Boolean**: the truth value of a statement (must be either True or False)

Data Type Conversion

- int(): float OR string with an integer inside → integer (removing any decimals)
    - E.g. int(10.9) → 10, int("10") → 10, but int("10.9") → error because the string has a float inside, not a direct integer
        - If necessary: convert the string to a float first, and then the float to an integer
    - Nothing is written inside the parentheses → 0
- float(): integer OR string with any number inside → float
    - E.g. float("10.9") → 10.9, float("10") → 10.0
    - Nothing is written inside the parentheses → 0.0
- str(): any value → string
    - E.g. str(10) → '10', str(10.0) → '10.0', but str(word) → error (if the word wasn't assigned a value!)
        - In the statement str(word), "word" represents a variable
    - Nothing is written inside the parentheses → ''

Built-in Math Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- ** Exponent
- % Modulo
    - Returns the remainder after division
- // Floor division (aka. integer division)
    - Refers to rounding down after division
    - Equivalent to math.floor(a/b)
    - a // b is not equivalent to int(a/b) when it comes to negative numbers, as -3 // 2 → -2, but int(-3 // 2) → -1

Notes

- Results aren't always precise

    - 2/3 + 1 → 1.6666666666666665 (note the 5, instead of how it's usually rounded to 7 in math)
- Order of operations apply to Python, including brackets and powers
    - 2**3**2 → 512, as 3**2 is calculated first
    - (2**3)**2 → 64, as 2**3 is bracketed and calculated first

String Concatenation

- Multiple strings added, or **concatenated** using the + operator → 1 large string
- A string can be multiplied by an integer using the * operator to repeat the string by an integer amount of times
    - E.g. "33" * 3 = "333333"

Print Function

- **print()**: built-in function used to output a string value
    - Values within parentheses are called **arguments**
    - No argument within the parentheses will return error
    - Arguments can be math operations, string concatenations, variables, etc.
- Values can be separated with a comma
    - In print(), separating values with commas (e.g., print("hello", "world")) automatically adds a space between them ("hello world"), unlike string concatenation ("hello" + "world") which does not add a space
    - By default, print() ends with a new line. You can change this using the end keyword
        - E.g. print("hello world", end="") will return "hello world" WITHOUT returning to a new line
    - The end keyword must come after all other values in the print() function
        - E.g. print("hello", "world", end="")
    - The end keyword must either not be entered or have a string following it (including empty strings ""), but not integers or floats

Input Function

- **input()** is a built-in function that allows the user to input a value themselves
    - This is often assigned to a variable, e.g. num = input('Enter a number: ')
    - If it is written without an assignment (e.g. input('Enter a number: ')), the data will not be saved and cannot be used later on

    - If no argument is given (e.g. input()), the user can simply click “enter”, and the value of the input (if assigned to a variable) will be nothing, despite the variable being valid
- The value returned by input() is always a string by default
    - This can be converted to different data types such as int() or float() under the correct conditions

Expressions

- Values and operators = expressions
- Evaluates down to a single value

Modules

- Built-in functions (like print(), input(), and abs()) are always available and can be used directly in any Python program
    - E.g. abs(-4) returns 4.
- Python also includes a standard library, which is a collection of modules
- Each **module** provides related functions that you can import and use in your programs
- Can be imported with the statement “import [module name]”
    - E.g. import math → gives access to the math module
        - math.pi → approximate value of Pi
        - math.ceil() → nearest integer ≥ number in parentheses
        - math.floor() → nearest integer ≤ number in parentheses
            - a // b is the same as math.floor(a/b) (hence the name floor division)
        - math.sqrt() → square root of the number in parentheses
    - E.g. import random → gives access to the random module
        - varName = random.randint(start, end) → assigns a random integer between the starting and ending integer, both inclusive

Indents

- **Indents** are large spaces at the beginning of a line, indicating a **block** of code
    - A **block** is a unit in coding
    - In Python, they are required while other coding languages only use indents for readability

Lists

- **List**: a collection of items in a particular order
- Indicated by square brackets []
- Different elements are separated by commas
- Can contain any value or data type
- Assigned to a variable
    - E.g. list1 = [1, 2, “hello”, “world”, True]
- **Index**: a unique position each element has
    - Starts from 0! Hence the first element of a list has the index 0, the second has the index 1, etc.
        - The last term of a list has the index -1, the second to last term -2, etc.
    - Used to access that specific element from a list with the code list[index]
        - E.g. using the list1 from earlier: list1[0] → 1, list1[3] → “world”
    - Modifications can be done with the code list[index] = new_value
        - E.g. using the same example: list1[1] = 3
        - list1 is now [1, 3, “hello”, “world”, True]
    - Swapping positions between elements of a list can be done using **multiple assignments**
        - E.g. list1[1], list1[3] = list1[3], list1[1] (meaning the 4th element is now the 2nd, and vice versa)
        - list1 is now [1, “world”, “hello”, 3, True] because the 2nd and 4th elements are swapped

List-Related Functions

- Always replace placeholders like list or list1 with your actual list names. E.g. list should be replaced with the list name
- len(list) → how many elements there are in the list
    - len(string/int/float) → how many characters/digits there are in the value
- list.index(value) → the index (position) of the first occurrence of the value in a list
    - str(value).index(value) → position in terms of characters (the x-th character, starting from 0) of the first occurrence of the value in a string

    - Not applicable to integers or floats
- list.append(value) → adds value to the back of the list
- list.insert(index, value) → adds value BEFORE the existing value of the index
- list.remove(value) → removes first existing occurrence of the value
- list.pop(index) → removes and returns the value at the given index
- list.sort(reverse=True/False) → sorts the list alphabetically and numerically with the option to do it in reverse order
- list1 = list.sorted(reverse=True/False) → makes a new, sorted version of the original list
- list1 = list.copy() → makes a **shallow copy** of the original list
    - **Shallow copy**: an exact replica of the original list, but the modification of one list will not affect the other
- list.reverse() → reverses a list's order of values
- list.clear() → clears all values of a list
- list.count(value) → the number of times a value appears in a list
- list.extend(list1) → extends a list by the values of another list at the end of the first list
- max(list) → maximum numerical value in a list
- min(list) → minimum numerical value in a list
- str("value").split() → creates a list with the elements inside the string, separated by a certain character (defaults to a space, but can be changed within the parentheses after split)
- sum(list) → sum of all numerical values in the list
- math.prod(list) → product of all numerical values in a list (note that the math module has to be imported)

For Loops

- A keyword that iterates/repeats a block of indented code for a specified amount of times
    - E.g. "for x in list:" or "for x in range(start, stop, step):" (note the colons!)
    - **range()**: a built-in function commonly used in loops, but it can also generate a sequence of numbers
        - **Start**: the integer the loop starts at (default is 0; can't be a decimal); inclusive
        - **Stop**: the integer the loop stops at (e.g. range(1,10) will stop when it reaches 10, but the value of 'i' stops at 9, as it starts from 0); exclusive

- **Step:** the number of times the range 'skips' numbers (default step is 1)
- Can be used to create lists including numbers within a range, e.g. varName = list(range(1,10)) # note that the word "list" is used intentionally this time!

If Statements

- Has three statements: if, elif, and else
- **if**: the indented code below is implemented only when the specified condition is true
- **elif**: similar to the if statement, but can only be used after an if statement is already used. However, if the if statement's conditional is already True, the indented code below this will be skipped.
- **else**: put at the very end of the "if-elif-else" conditional chain. This statement ONLY runs if the conditionals for all the if and elif statements are all False.

Relational Operators

- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to
- == Equal to
- != Not equal to

List Comprehension

- Special syntax to create a list in a single line
- list = [variable for variable in iterable if condition]
- e.g. list = [i for i in range(1,10) if i % 2 == 0]
    - Creates a list of every integer (denoted i in this case) from 1 to 9 (since 10 is excluded) which satisfies the if condition i % 2 == 0

Boolean Operators

- Has only two values: **True** or **False**
    - E.g. 8 == 8 → True, 39847 == 34 → False, "67" == 67 → False (since they are two different data types)
- Values are assigned a truth value
    - The Boolean value of 'False' returns False

- 0, 0.0, or None → False
- Empty data types/values, sets and lists → False
  - “” (empty string), [] (empty list), etc.
- Other values mostly return True

- Operators include:
  - **and**: requires the specified two conditions to BOTH be True for the entire conditional to be True.
  - **or**: requires at least one of the specified two conditions to be True for the entire conditional to be True. Note: If the first of the specified two conditions is read to be true, the computer skips over the second condition regardless of whether it results in an error. This is known as short-circuiting.
  - **not**: reverses the conditional’s statement; if a condition is False, it will reverse it and output True
  - **not** is evaluated before **and**; **and** is evaluated before **or**:
    - The priority can be illustrated in brackets with the example ((not 10 < 9) and (10 < 11)) or (“hi” == “hi”)

Expression	Evaluates to
True and True	True
True and False	False
False and True	False
False and False	False

Expression	Evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

Expression	Evaluates to
not True	False
not False	True

Precedence of Operators

- In the order of first to last priority (true unless otherwise bracketed in a code):
  - Exponentiation
  - Multiplication/Division/Modulo/Floor Division (Integer Division)

- Addition/Subtraction
- Relational Operators
- Boolean Operators

While Loop

- Used when the number of iterations of a statement is an uncertain amount
- Syntax: while [condition]: [statements in a different indented line]
  - **Infinite loop:** a loop that never breaks and repeats the statement forever (an error)

Augmented Assignment Statement

- x += y means x = x + y
- x -= y means x = x - y
- x *= y means x = x*y
- x /= y means x = x/y
- x %= y means x = x%y
- x //=y means x = x//y

String Manipulation

1. Strip Function

- strName.strip('object') → returns a copy of the string with all characters in 'object' removed from both the beginning and end (default removes whitespace such as spaces, tabs, or newlines)
- Example: 'abbacdbba'.strip('ab') → 'cd'
  (All 'a' and 'b' characters at both ends are removed; order doesn't matter)
- Variations:
  - strName.lstrip('object') → removes specified characters only from the beginning
  - strName.rstrip('object') → removes specified characters only from the end

2. String Indexing

- Similar logic to indexing a list, but instead of separate elements, each character in a string has its own index
  - E.g. in string st = 'abcd', st[0] → 'a', st[1] → 'b', etc.
  - As such, strName.index('character/s') also works as it does to a list — for strings, this is usually 1 character, but can be 2 grouped together, which will be considered one single index
  - len(strName) also works to return the number of characters in a string

3. String Slicing

- strName[start (inclusive): end (exclusive)] → every character of the string from strName[start] all the way to strName[end-1] (since it's exclusive)
    - If indices are out of range or the range is invalid, an empty string '' is produced instead (NOT an error)

4. Membership Operators

- "in" is known as the membership operator; a statement with "in" is a Boolean
    - E.g. 'abc' in 'abcde' → True (can also be used with the "if" statement)
    - "not in" → opposite of "in"

5. Join and Split Functions

- delimiter.join('string'): separates the contents of the list/string using delimiter
    - Ex: print('-'.join('hello')) → "h-e-l-l-o"
- Opposite of the split() function

6. Character Case Functions

- upper(), lower(), & title() functions:
    - upper() capitalizes all the characters in the string
    - lower() function lowercases all of them
    - title() function capitalizes the first character after a space.
        - Note: special characters such as "!", "?", spaces, etc. are unchanged.
- There are also ways to check if the string is completely capitalized, lowercase, etc. through the functions isupper(), islower() & istitle().
    - These functions return True if the characters in the string are capitalized, lowercase, and if only the first character after a space is capitalized, respectively.

7. Checking String Properties

- Lastly, there are also functions to determine the character types of an index of a string. The three functions are isdigit(), isalpha(), & isalnum(). These functions return either True or False.
    - isdigit() checks if all the characters in the string are numbers, or digits.
    - isalpha() just checks if all the characters in the string are alphabets.
    - isalnum(), checks if all the characters are alphabets or digits.