# IDX G9 Programming Essentials H
# Study Guide Issue Semester 1 Finals
# By Loren and Forrest, Edited by Cathy



**Since the scope for the final exam is everything learnt this semester, please also check the previous issues while reviewing.**

**Contents:**



## Dictionary

- Dictionaries store data in key-value pairs indicated through braces {}, where each key is assigned a corresponding value.
    - Keys and values can be any data type, but keys must be immutable (e.g. strings, numbers, tuples) and cannot repeat, while values can be of any type and may repeat.
        - If a key is repeated, the later value will overwrite the earlier one.
    - A dictionary is unordered and does not use numeric indices, so you cannot access its elements by position or slice it like a list.
    - Create a new empty dictionary with dic = dict() or dic = {}
- A key is added through assignment (with the equal = sign) rather than functions like append() or add().

    - For a dictionary “dictName”, the key “keyName” can be added or modified with a corresponding value “value” through dictName[keyName] = value.
        - If keyName already exists, its value is modified; if it does not exist, a new key-value pair is created.
    - In the dictionary, pairs will appear separated by commas, with each key and value separated by colons, such as {keyName: value, keyName1: value1}.
- A dictionary’s keys and values can be accessed through dictName.keys() and dictName.values() respectively, and its key-value pairs within tuples can be accessed through dictName.items().
    - For a dictionary dictName = {keyName: value, keyName1: value1}**,** list(dictName.keys()) would return [keyName, keyName1] (with a similar logic for .values() as well).
    - However, list(dictName.items()) would return [(keyName, value), (keyName1, value1)]; note the tuples within the list.
        - Example: For dictName = {'apple': 2, 'banana': 5}, dictName.keys() returns ['apple', 'banana'], dictName.values() returns [2, 5], and dictName.items() returns [('apple', 2), ('banana', 5)]

          **(shown as lists for clarity, need list() function)**.
        - The list() function is used for the keys and values to be returned properly. Without it, they would be returned as dict_values(<insert content here>) which is not a subscriptable datatype, meaning it cannot be sliced and is generally difficult to work with.
- A dictionary’s key-value pairs can be removed by deleting its key with the del keyword.
    - For a dictionary dictName = {keyName: value, keyName1: value1}**,** using del dictName[keyName] deletes the (keyName, value) pair altogether.
- Dictionaries can be nested:
    - A list can contain dictionaries
    - A dictionary can contain lists
    - A dictionary can contain another dictionary
- Assigning one dictionary variable to another creates a reference, not a copy.
    - To create a shallow copy, use dictName.copy() or dict(dictName).

**Comments**

- Comments are added through the hashtag sign #<content> or a pair of triple single quotation marks ‘‘‘<content>’’’.
    - All characters after the hash sign (#) up to the end of the line are part of the comment and are ignored by the Python interpreter.
    - Comments written with triple single quotation marks are commonly used as multiline comments
- Comments are used to improve code readability, make the code easier to modify, and help developers keep track of the code’s working status.
- Docstrings are a type of comment specific to a function, added through a pair of triple single or double quotation marks “““<docstring>”””.
    - Docstrings are written to explain what the function does without readers having to go through its entire code.
    - A function’s docstrings can be accessed through functionName.___doc___ or help(functionName).
        - While functionName.___doc___ returns only the docstring, help(functionName) returns the function itself (along with its parameters!) with its docstring in indents (for reference, try help(help()) in python IDLE.)

**Files and File Paths**

- Every file has a filename and a path.
    - The filename is the specific name of the file, which comes with an extension indicating the filetype (such as how pyFile.py is a filename and .py is its extension).
    - The path is a sequence of folders indicating where the file is located in (for example, C:\Users\Name\pyFolder\pyFile.py is the full path of pyFile.py; note that the filename is included within the path).
        - The term folder is interchangeable with the term directory.
- All paths have a root folder.
    - On windows, this folder is a drive such as C:\, while on Linus and OS X it is a forward flash /.
        - Note the backslash \ for windows which is used as the path separator, accessed through os.path.join(‘multiple’, ‘directories’, ‘here’) after importing

the os module, resulting in the output ‘multiple\\directories\\here’. The double backslash appears because Python escapes the backslash character internally.

        - For OS and Linus, the forward slash is used instead as the separator.

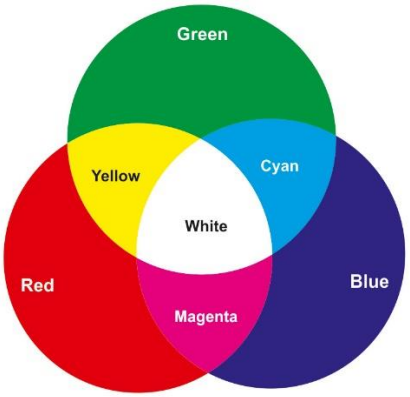
- New volumes appear as folders under root directories, such as D:\ for another drive on Windows, /Volumes on macOS, or /mnt on Linux.
- Different parts of a path can be accessed with different functions.
    - Using os.path.dirname(path) after importing the os module returns a string of everything before the last slash in a path. With the example earlier, it returns ‘C:\Users\Name\pyFolder’.
    - Using os.path.basename(path) returns everything after the last slash, such as ‘pyFile.py’ for the example earlier.
    - Using os.path.split(path) returns both values as a tuple, such as (‘C:\Users\Name\pyFolder’, ‘pyFile.py’) for the example earlier.
        - For windows users!!! It is recommended to use os.path.split(r‘<path>’), note the letter r before the path, rather than just (‘<path>’) because windows uses the backslash \ as the separator while backlashes are also used in python for escape sequences (meaning things like \n, \t, etc.) and it will likely return a syntax error.
    - For any path assigned to a variable named filePath, using filePath.split(os.path.sep) returns every component of the path between slashes. For the example earlier, it returns [‘C:’, ‘Users’, ‘Name’, ‘pyFolder’, ‘pyFile.py’]
        - For OS X and Linus users, there will be an empty string ‘’ at the beginning of the list because the root folder is always a forward slash / which is also its separator.
- The current working directory (or cwd) is the directory that the program is currently in, obtained through os.getcwd() and changed through os.chdir(path).
    - Changing the directory to one that doesn’t exist returns an error.
        - Whether a path exists can be checked through os.path.exists(path), returning a Boolean value (True or False).
        - Whether a path is a file can be checked through os.path.isfile(path).
        - Whether it is a folder can be checked through os.path.isfolder(path).

- A path can be specified through either a relative or absolute path.
    - An absolute path begins with the root folder and covers every folder all the way until the file.
        - Whether a path is absolute can be checked through os.path.isabs(path).
        - Converting a path to one can be done through os.path.abspath(path).
    - A relative path is relative to the program's cwd.
        - Using a single dot . represents the program's full cwd, while .. represents the cwd's parent directory, such as how in C:\parentDir\dir, parentDir is the parent directory of dir.
        - Using the single dot is optional. The lines ./pyFile.py and pyFile.py are both relative and refer to the same file.
        - Converting a path to a relative one can be done through os.path.relpath(path, start), returning a relative path from start to path as entered, with start being the cwd by default.
        - os.path.relpath('C:\\Windows', 'C:\\dir1\\dir2') returns '..\\..\\Windows', as to get from dir2 to dir1 (its parent directory), two dots .. are needed, then from dir2 to Windows (its parent directory), two dots are needed again.
- New folders are created with os.makedirs(path).
    - Multiple folders within that folder can also be created at the same time, meaning the function creates any immediate folders necessary to ensure the full path exists.
    - For example, in a computer where no folders exist, os.makedirs('C:\\newFolder1\\newFolder2\\newFolder3') creates all 3 entered folders.
- The size of a file in bytes can be accessed through os.path.getsize(path).
- All filenames within a directory can be accessed through os.listdir(path).
- Files can be copied using shutil.copy(source, destination) after importing the shutil module, and returns a string of the absolute path of the copied file**.**
- Folders and their contents can be copied using shutil.copytree(source, destination), returning a string of the absolute path of the copied folder.
- Files or folders can be moved using shutil.move(source, destination), returning a string the absolute path of the new destination of the file or folder.
    - If the destination is a folder, the source will be moved to the folder.

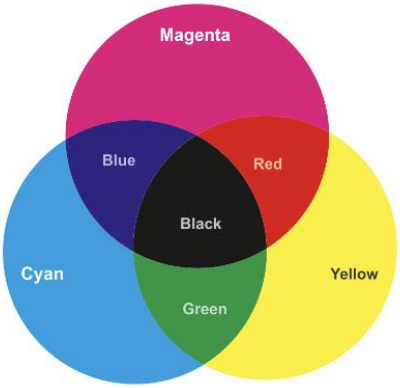
        - If that folder already has the file with the same name as the source, it will be overwritten.
        - If the folder doesn't exist, then the source will be renamed the name of the non-existent folder.
    - If the destination is a filename, the source will be renamed.
    - The destination can also be a different folder and a different filename, renaming and moving the source at the same time.
        - If the folder doesn't exist, python will return an error. In this case it will not rename the source because the user already specified a filename within the folder.
- Deleting a file can be done with os.unlink(path).
- Deleting an empty folder can be done with os.rmtree(path).
- Deleting a folder and its contents can be done with shutil.rmtree(path).

**Computer Colors**

- All colors are generated from three spectral hues of red, green, and blue (RGB).
- The proper reference for printing color is cyan, magenta, yellow, and key (black), or CMYK.



RGB - Red, Green, Blue

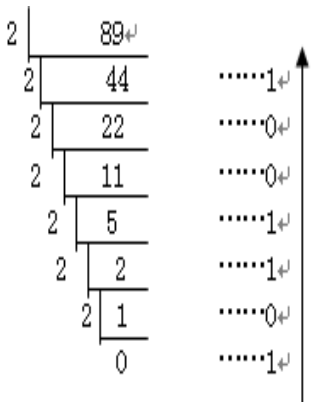


CMYK - Cyan, Magenta, Yellow, Black

- Colors can be identified using three decimal numbers from 0–255, each representing the intensity of red, green, and blue respectively.
    - These values are usually written in the form (R, G, B), for example (255, 0, 0) represents red.
    - The range 0–255 gives 256 possible levels for each color channel.
- Colors can also be represented in binary form (0 or 1), 24 bits are used for representation.

- Colors can also be represented using hexadecimal notation.
    - In hexadecimal color codes, a color is written as #RRGGBB.For example, #FFFFFF represents white, where each pair corresponds to red, green, and blue.

**Number System**

- The base of a number is how many digits that number system uses to represent a value. The base is written in subscript after the number itself, such as $number_{base}$.
- The denary/decimal system uses 10 digits to represent numbers and is thus base 10. In this case, the base may be omitted from the number (e.g. 3984 = $3984_{10}$).
- The binary system is most commonly used in computer science, using 2 digits (0 and 1).
    - Each 0 and 1 is called a bit (binary digit), with the unit b.
    - A group of 8 bits is called a byte, with the unit B (uppercase), representing storage space.
- $abcd_{base} = d*base^0 + c*base^1 + b*base^2 + a*base^1$, with this same logic extending for any number of digits for any base.
- $abcd_{10}$ is converted to another base by taking the reverse order of the remainder of abcd divided by the base, then floor dividing abcd by the base, then taking the remainder of that number divided by the base until that number becomes floor divided into zero.
    - E.g.: decimal to binary
    - 

- The hexadecimal system is based on 16, where numbers beyond 9 will translate into alphabets in ascending order (10 becomes A, 11 becomes B, …)
    - Each hexadecimal digit is 4 bits.
- Binary can be added, subtracted, multiplied and divided like decimal numbers can.
    - 0+0=0; 0+1=1; 1+0=1; 1+1=0 carry 1

```
  1 1 1 1 1     (carried digits)
    0 1 1 0 1
+   1 0 1 1 1
-------------
= 1 0 0 1 0 0 = 36
```

- 0-0=0, 0-1=1 borrow 1, 1-0=1, 1-1=0

```
   *   * * *   (starred columns are borrowed from)
  1 1 0 1 1 1 0
-     1 0 1 1 1
---------------
= 1 0 1 0 1 1 1
```

- Bitwise operations used in binary are like logical operators (not, and, or). They are represented through & (AND), | (OR), and ^ (XOR), in the same order of priority as logical operators where XOR>AND>OR.
  - The XOR value of two numbers is 1 if they are different (e.g. 0^1 == 1) and 0 if they are the same (e.g. 0^0 == 0).
  - The AND value of two numbers if 1 if and only if both of them are 1, and 0 otherwise (e.g. 1&1 == 1, while 0&0 == 0 and 1&0 == 0).
  - The OR value of two numbers is 1 if one or more of them are 1, and 0 otherwise (e.g. 1|1 == 1, 1|0 == 1, 0|0 == 0).
  - These operations may also be used with other bases like decimal, but should be first converted to binary then have the value converted back to the preferred base.