Exam Material 1

Week 1 & 2Storage 
Devices 
e.g., Disk, CD, 
and Tape 
Memory 
Bus 
Communication 
Devices 
e.g., Modem, 
and NIC 
Input 
Devices 
e.g., Keyboard, 
Mouse 
Devices 
e.g., Monitor, 
Printer 

CPU - central processing unit - brain - volatile

Memory - RAM - random access memory - volatile

Storage - HDD Hard disk drive, SSD solid state drive - nonvolatile

Volatile - if you turn off your computer, information stored will be lost

CPU is like knowledge in your brain and Memory is like knowledge in your notes or a textbook on your work desk in terms of how much you can store and how quickly you can access that knowledge. Storage is the bookshelf - hold a ton of information but accessing it is incredibly slow; to use it, you have to bring it to the work desk.

b is for bit

B is for byte - byte is 8 bits

How to count in binary

Visual: <https://i.imgur.com/NQPrUsI.gifv>

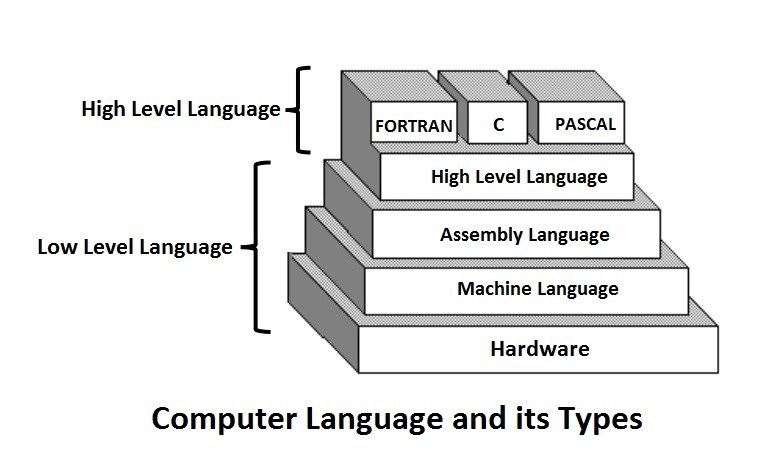
Math way: <https://www.youtube.com/watch?v=a2FpnU9Mm3E>

Symbol Terminology

|  |  |
| --- | --- |
| **‘** | Quote, single quote |
| **“** | Quote, double quote |
| **[ ]** | Bracket, square bracket |
| **{ }** | Brace, curly brace |
| **( )** | Parentheses |
| **/** | Slash |
| **\** | Back slash |
| **:** | Colon |
| **;** | Semi-colon |
| **#** | Pound, hashtag |
| **^** | Caret, up caret |
| **< >** | Angle brackets, side carets, less than/greater than |
| **&** | And, ampersand |
| **|** | Pipe |

= can be called the equal sign, but it’s **formally** known as the assignment operator

Language Hierarchy

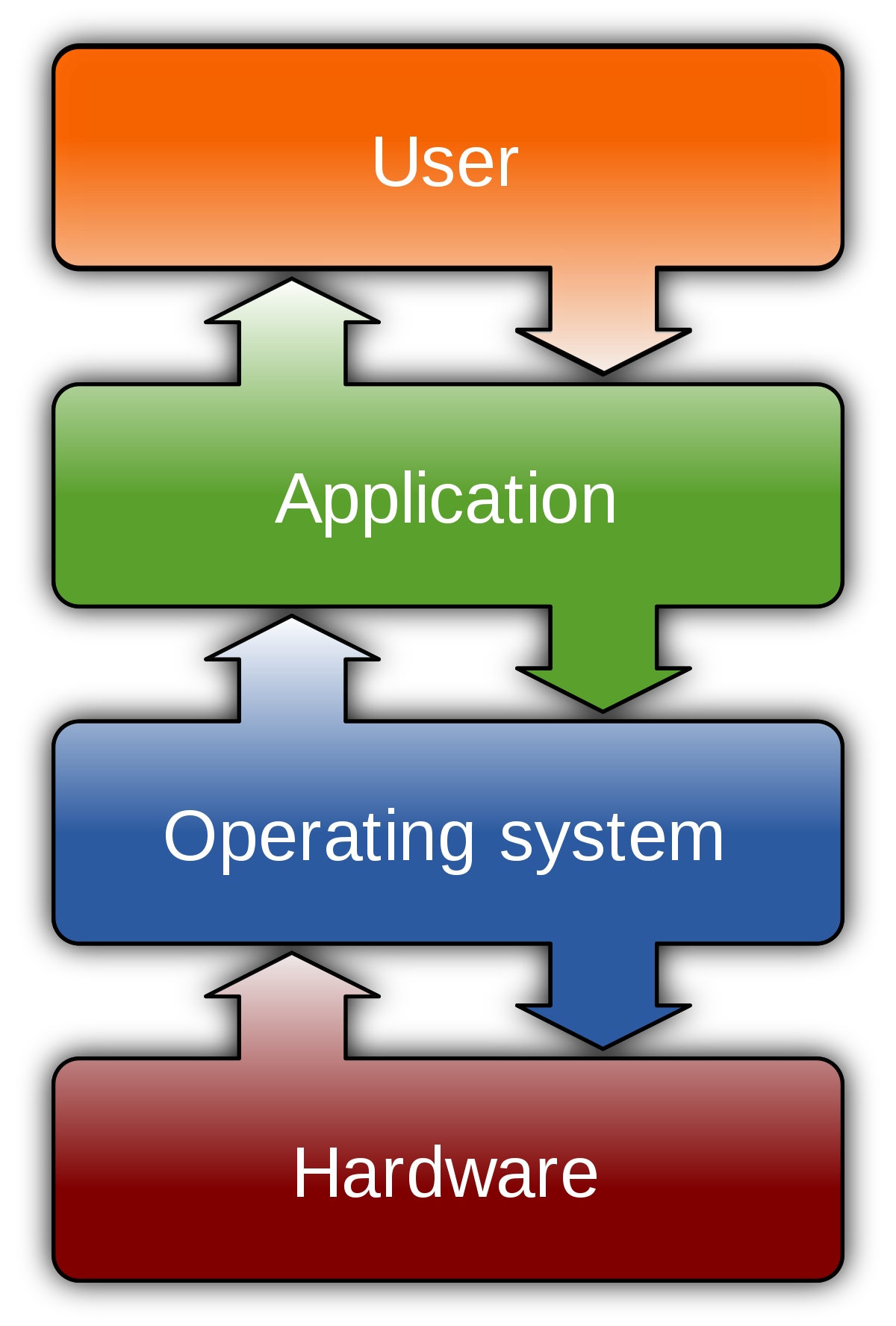


Interpreted Language vs Compiled Language

* Interpreted means it reads line by line as it runs - requires an interpreter be installed
* Compiled means it reads the whole file before it runs - makes an additional file to be ran

3 Major Tasks of the Operating System - Section 1.4 in Textbook

1. Controlling and Monitoring System Activities
2. Allocating and Assigning System Resources
3. Scheduling Operations



Data Types (learn more later)

* String - str - from single letter to multiple words - in quotes or double quotes - some languages care, but Python doesn’t. Good habit is single quotes for single letters and double quotes for everything else
* Number
  + Integer - int - whole number
  + Float - number with decimal
* Variable - named item that can hold many types of data
* Function - named set of code that can take parameters/arguments, does “stuff”, and can return a value (more details on functions later on)

Rules and Conventions

Rules - stuff you have to follow to get your code to run

* Strings must have quotes or double quotes around the words
* Comments begin with # or are surrounded by three single quotes (multiline)
* Everything is case sensitive so be mindful

Conventions - stuff you follow to make your code readable and pretty

* Comment on the top of every file
  + # Name
  + # CS 1400
  + # Assn # - Task #
* Add comments throughout the code to tell anybody reading you code what’s going on
* Add blank lines to separate code into “paragraphs”
* Variable names
  + Use descriptive variable names - full words
  + lowerCamelCase
  + No single letters
  + Should be nouns
* Use a space on each side of an arithmetic symbol

3 Types of Errors

1. Syntax Error - Program won’t even run
2. RunTime Error - Program will crash in the middle
3. Logic Error - Program runs, but produces wrong answer

Functions

* print() - anything you put in the parentheses gets printed to the console
* input() - put a prompt in the parentheses, and will wait for the user to give input and hit enter - returns a string
* eval() - put a numeric string in the parentheses, and it will return the numeric value
  + Best practice to call eval() on every numeric input you expect
  + Don’t call it on inputs you want to remain as strings (names, phrases, etc)

Turtle module - <https://docs.python.org/3/library/turtle.html>

* You import modules with import module
* Allows you to use all the functions within that module
  + module.function()
* Most important functions for this class
  + turtle.speed(0) - makes the turtle run at the specified speed - 0 is nearly instant
  + turtle.forward(numPixels) - goes forward n amount of pixels
  + turtle.left(degrees) & turtle.right(degrees) - turns the direction of the turtle by x degrees
  + turtle.goto(x, y) - turtle travels to the x, y coordinates given
  + turtle.setheading(degrees) - sets turtle direction on 360 scale - 90 is up, 180 is left, etc.
  + turtle.penup() & turtle.pendown() - to use when you want to move but not draw
  + turtle.circle(radius) - draws a circle (starting from the bottom of the circle)
  + turtle.color(“color”) - changes turtle to specified color
  + turtle.begin\_fill() & turtle.end\_fill() - will fill shape currently being drawn - ms paint bucket
  + turtle.done() - put at the end of the file to make the turtle screen not disappear

3 Types of Assignment

* Multiple assignment - multiple equal signs
  + x = y = z = 1
* Self assignment
  + x = x + 1
  + On the right side, use the original value of x
* Simultaneous assignment
  + one, two = 1, 2

Week 3

Operators

Two general types

* Binary Operator - binary means two - an operator that uses two operands
  + + addition
  + - subtraction
  + \* multiply
  + / divide
  + // integer division
  + \*\* exponentiation
  + % mod/modulo/modulus - the remainder operator
* Unary Operator - operator that uses one operand
  + + positive - just put it at the front of the number without a space
  + - negative

Order of operations

1. ()
2. +, - (unary (when it makes a number positive or negative))
3. \*\*
4. \*, /, //, %
5. +, - (binary)

Augmented operators

Overall format: variable <operator>= value

variable = variable <operator> value ← identical in functionality to line above

* +=
* -=
* \*=
* /=
* //=
* %=
* \*\*=

Additional use for +

* Concatenation operator
  + Can ‘add’ strings together
  + print(“Hello” + “world”)
    - Prints: Helloworld
    - Note: you have to add spaces manually this way
  + Must convert numbers to strings to concatenate
  + print(“Hours: ” + str(10))
    - Prints: Hours: 10

Time Module

* import time
* For the whole semester, you’ll only use time.time() and time.sleep()
* time.time() returns the amount of seconds since Jan 1, 1970, aka the current time

Rounding

* Python uses banking rounding
  + Rounds to the nearest even number when number ends in .5 exactly
* int() returns the integer portion, truncating the rest
* round(number[, numDigits]) - brackets mean optional in parameter
  + round(15.66666) = 16
  + round(15.66666, 2) = 15.67

Software Development Process - Plan your software early

1. Requirement Specification
   1. Document written in English
   2. Seeks to understand the problem
   3. Users and developers are involved in making it
   4. Detailed as possible
2. System Analysis
   1. Concerned with the flow of data
   2. Formulas - how we process the data
3. System Design
   1. Developers write this in English
   2. Blueprint for the software
   3. Sometimes called “Pseudocode”
4. Implementation
   1. Code
   2. Test
   3. Debug
5. Testing
   1. Should have at least 2 Test cases
      1. Specify input
      2. Expected output
6. Deployment
   1. Software release
7. Maintenance
   1. Add features and Update
   2. Fix newly discovered errors

Built-in functions

* print()
* input()
* int()
* float()
* str()
* abs() returns absolute value
* min() returns minimum value in a list of values
* max() returns maximum value in a list of values
* round() rounds a number
* pow(a, b) same as a \*\* b

Week 4

Always import modules at the top of your py file - only comments should go above it

Two main ways to import modules

* import module import math math.sqrt()
* from module import function from math import sqrt sqrt()

ASCII

* 7-bit encoding scheme
  + Values of 0-127 - 128 possible values

UTF-16

* 16 digits - 0-9 and A-F
* Hexadecimal

Functions for characters

* ord(x)
  + ordinal
  + Returns the ascii number of a character
* chr(x)
  + Returns the character where x is an ascii number

Escape sequences

* \u denotes next few letters/numbers are unicode
* \” allows double quotes to be in string - same idea works for single quotes
* \t adds a tab to a string
* \n adds new line
* \\ allows backslash to be in string

Use two print statements to print on only one line

* Add a last parameter to the print function

print(“Hello”, end=””)

print(“World”)

Functions you can call on all objects - strings are objects, etc.

myObject = …

* type(myObject) returns the data type - ie str, int, etc
* id(myObject) returns the unique number of the object

Functions you can call on strings - use the access operator (the dot) on it

myString = “Whatever”

* myString.lower() whatever
* myString.upper() WHATEVER
* myString.isdigit() returns true or false, if the string is a number

format() function takes 2 parameters separated by a comma

1. The string or number you want to format
2. The format string (see below)

How to create a format string for the format() function

* ^, >, < Center, Right align, Left align
* # How wide you want the formatted area in the console
* .# Number of decimals
* f, s Float, string

Exam Material 2

Week 5

Boolean - data type that can either be True or False

Expressions that evaluate to True or False

* Comparison Operators
  + < less than
  + > greater than
  + <= less than or equal to
  + >= greater than or equal to
  + == equal to (use two ‘=’ when you test equality)
  + != not equal to

random module

Important functions in this module

* random() returns float from [0.0, 1.0) (including 0 but not 1)
* randint(a, b) returns integer from [a, b] (including both a and b)
* seed(x) tells the computer to stop being random afterwards

Random number generation on computers is NEVER truly random

Conditional statements

Execute certain code if the condition is true

Broadly speaking, all of these can be referred to as if-statements

Indentation is everything - be sure to indent all statements properly

* if statement
  + One way condition
  + Syntax

if <condition>**:**

<statements>

…

<next code not part of if-statement>

* if-else statement
  + Executes one of two conditions
  + Syntax

if <condition>:

<statement A>

…

else:

<statement B>

…

<next code not part of if-else-statement>

* Nested if-statement
  + if-statement within an if-statement
* if-else-if statement
  + Executes only one of many conditions - only executes the first that is true from top to bottom
  + Syntax

if <condition A>:

<statement A>

...

elif <condition B>:

<statement B>

…

elif <condition C>:

<statement C>

…

…

else:

<statement Z>

…

<next code not part of if-else-if-statement>

Simplifying Boolean expressions

1. answer = True

if answer == True:

<whatever>

else:

<whatever>

Can be simplified to:

if answer:

<whatever>

else:

<whatever>

1. number = 10

if number % 2 == 0:

even = True

else:

even = False

Can be simplified to:

even = number % 2 == 0

Week 6

Logical Operators - aka Boolean operators

* not logical negation
  + changes boolean from one value to the other
* and logical conjunction
  + Returns true if both conditions are true
* or logical
  + Returns true if either condition is true

\* Important note, both sides of an “and” or “or” need to be complete statements

if dog == black or yellow # will not work

if dog == black or dog == yellow # will work

Truth Tables - <https://introcs.cs.princeton.edu/java/71boolean/images/truth-table.png>

Short Circuiting

Under certain circumstances, you don’t need to evaluate the second because of the evaluated value of the first

Order of operations

1. ()
2. +, - (unary (when it makes a number positive or negative))
3. \*\*
4. not
5. \*, /, //, %
6. +, - (binary)
7. <, <=, >, >=
8. ==, !=
9. and
10. or
11. =, +=, -=, (other augmented operators)

How to change variable name

Right click variable -> refactor -> rename

Import statements should always be at the top of the files - only comments above

How to terminate program

import sys

sys.exit(<num>)

Will print “Process finished with exit code <num>”

How to do if-statement in one line

<result 1> if <condition> else <result 2>

Loops!

loop• 
F alse 
continuation 
condition? 
Statement(s) 
f ho bodvj 
count = 
count < 100? 
IS fun!" ) 
count = count + I 
False 

While-loops

while <condition>:

<statement to repeat>

…

<statement outside loop>

* Common errors
  + Off-by-one error
    - Typically happens when you use the wrong comparison operator
  + Infinite loop
* Ways to control While-loops
  + Counter variable
    - while count < #:

count += 1

* + User-controlled value
    - while continue == “Y”:

continue = input(“Enter Y to loop”)

* + Boolean/Sentinel value
    - while not isDone:

isDone = True

For-loops

for <var> in range(<range>):

<statement to repeat>

…

<statement outside loop>

* How to use range()
  + range(a) a sequence of numbers [0, a)
  + range(a, b) a sequence of numbers [a, b)
  + range(a, b, s) a sequence of numbers [a, b) but only every ‘s’th number

Break and Continue

* continue will skip the rest of the loop body and go onto the next number
* break will end the loop entirely

Week 7

Functions

Similar to loops, but will only run when you ask it to - like a fire evacuation plan

Write code that can be used later

Syntax to set up

def <functionName> (<formal parameters>): # optional to have parameters

<statements>

...

return <return value> # optional to return something

# if you don’t return something, it returns None

Purpose of having functions

* Reuse code
* Code organization
* Simplify code

How to Use

* “Call” a function
* Be careful on the parameters - they don’t have to have the same name, but they do have to be in the same order in most cases
  + Parameter can also be called arguments
* You can call functions within other functions, but you should NOT define functions within other functions
* If a function returns something, be sure to save it as a variable
  + value = valueFunction()

Positional parameters vs. Keyword arguments

* Positional: If you put all the actual parameters in the same order as the formal parameters in a function call
* Keyword: When you specify which actual parameter corresponds to which formal parameters in a function call. They don’t need to be in the same order. If you specify positional and keyword parameters, you must put all the keywords on the far right.

Default values

* In the function definition’s formal parameters
* formalParameter = defaultValue

How to create and use your own module

* Create a new .py file called fileName
* Put functions in fileName.py
* When you want to use it, import fileName or from fileName import function
* Use functions as fileName.function() or function()
* To import a module from a different folder import folderName.fileName

Returning multiple things from a function

* Possible in Python because of simultaneous assignment
* Put a comma between the variables you want to return

return val1, val2, val3

num1, num2, num3 = func()

Python is pass by value

Week 8

id() Each object has a unique id number

Number and string objects are immutable which means they can’t change

Global vs Local Variables

* Global variables are variables created in the file but not in a function
  + Scope: Can be used throughout the entire file
  + If you want to use a global within a function, put ‘global’ in front of the name
* Local variables are variables created in a function
  + Scope: Can only be used in that function

Exam Material 3

Week 9

Objects

Use a Class to create an Object Use a blueprint to create a house

An object is an instance of a class A house is an instance of a blueprint

Purpose of using objects

* Reuse code
* Code organization
* Simplify code

Functions vs Methods

* Methods are functions within a class
* Usage/calling:
  + Function: myFunction(param1, param2)
  + Method: myObject.myMethod(param1, param2)

Why do we use classes and objects?

* The data is bound to the object
  + aka, you don’t have to pass around function parameters because all the functions/methods in the class can share the same variables among themselves

Syntax:

class <ClassName>:

<initializer method>

<other methods>

…

Classes should only contain methods, and those methods should contain code

Classes should be in their own file / own module

typically filename - lowerCamelCase classname - UpperCamelCase

Syntax - initializer method

def \_\_init\_\_(self, params that you need to create said object):

…

Double underscore on front and back of method name = dunder method

Also called magic methods because they get called automatically

self

* A reference to the object itself (hence ‘self’)
* Variables that start with ‘self.’ can be used by any method in the class
* All methods in the class have ‘self’ as the first parameter
* Ignore ‘self’ outside of the class

How to access the variables and methods in a class inside of the class

1. Inside a method, if you want to use a variable self.variable
2. Inside a method, if you want to call another method self.method(<params>)

How to access the variables and methods in a class outside of the class

1. Make an instance of a class myObject = ClassName(<params>)
2. Use the access operator myObject.variable myObject.method(<params>)

Constructor the thing that actually creates an object

from fileName import ClassName

objectName = ClassName(params listed in \_\_init\_\_())

*Highlighted portions are the constructor*

UML - Unified Modeling Language

|  |
| --- |
| ClassName |
| All variables in the class that start with self. in the format of  variableName : type  ie radius : int |
| All methods/functions in the class in the form of  methodName(<param> : <type>, <param> : <type>, … ) : returnType  \* don’t include self as a parameter |

Private vs Public variables and methods

* Public variables and methods can be accessed outside and inside of the class
* Private variables and methods can only be accessed inside the class
* Public variable and method names look normal
* Private variable and method names begin with double underscore
* self.something and self.\_\_something are not the same
  + One exception to this:
  + methods that have double underscore on both side of the name are public
  + ie def \_\_init\_\_(): is public

Getters and Setters

* Methods that’s whole purpose is to either “get” or return a private variable or

“set” a private variable to a new value

Encapsulation and Abstraction

Week 10

Strings Revisited

Functions on strings

* len(myString) returns how many characters myString has
* max(myString) returns max based on ascii values
* min(myString) returns min based on ascii values

Methods on strings

* myString.lower() makes string all lowercase
* myString.upper() makes string all uppercase
* myString.isupper() returns true or false if string is uppercase
* myString.islower() returns true or false if string is lowercase
* myString.capitalize() returns string with only first letter uppercase

Operators on strings

* + concatenation add strings together
* \* repeat operator repeat the string # amount of times
* [<index>] index operator returns character at that index

Index of strings start at 0

* [<A>:<B>] slice operator returns a string from indexA to indexB - 1
  + [:num] will start at beginning and go to num-1
  + [num:] will start at num and go to end
* in contains operator returns true or false if substring is in string

Comparisons on strings - Checks ascii number/value one character at a time

* == equality
* <, <=, >, >= less than, less than or equal to, greater than, greater than or equal

Interning

* When two variables point to the same object in memory

Iterating

* Use a for-loop to go one letter at a time

for char in myString: # by Item way

print(char)

for i in range(len(myString)): # by Index way

print(myString[i])

Operator Overloading

Python knows how to do a lot of things with operators by default

* Adding two numbers together will produce a number
* Adding two strings together will produce a string
* But if you create your own object, like a Circle class, Python doesn’t know how to add two Circles together, so you have to tell it how by overloading that operator.

Operators to overload

Operators that return a value (string, int, float)

+ def \_\_add\_\_(self, other):

- def \_\_sub\_\_(self, other):

\* def \_\_mul\_\_(self, other):

/ def \_\_truediv\_\_(self, other):

// def \_\_floordiv\_\_(self, other):

% def \_\_mod\_\_(self, other):

Operators that return a boolean (true/false)

< def \_\_lt\_\_(self, other): # used for min()

<= def \_\_le\_\_(self, other):

> def \_\_gt\_\_(self, other): # used for max()

>= def \_\_ge\_\_(self, other):

== def \_\_eq\_\_(self, other):

!= def \_\_ne\_\_(self, other):

Operators that return a value (string, int, float) but only take one operand (hence, no ‘other’)

Print an object def \_\_str\_\_(self):

Similar to str def \_\_repr\_\_(self): # stands for representation

len() def \_\_len\_\_(self): # must return int (not float)

Week 11

Lists

A list is an object that contains other objects

Multiple ways to create lists

* myList = list() [ ]
* myList = list([‘stuff’, ‘other’]) [‘stuff’, ‘other’]
* myList = list(range(1, 6)) [1, 2, 3, 4, 5]
* myList = list(‘apple’) [‘a’, ‘p’, ‘p’, ‘l’, ‘e’]
* myList = [ ] [ ]
* myList = [0] \* 3 [0, 0, 0]

Lists are mutable

Lists begin at index 0

Methods on Lists - these modify the data directly, so you don’t have to assign them to a variable

* myList.append(<object>) Adds object to the end of the list
* myList.index(<object>) Returns the index number of where the object is
* myList.pop() Returns and removes the last object in the list
* myList.pop(<index>) Returns and removes the object at <index>
* myList.remove(<object>) Removes the first occurrence of <object>
* myList.reverse() Reverses the order of the list
* myList.sort() Sorts items in the list in ascending order

[] index operator

* Returns the item at a specific index

[:] slice operator

* Returns a list values between the first and second index

Functions on lists

* len(myList) returns how many items are in the list
* min(myList) returns lowest item in list
* max(myList) returns largest item in list
* sum(myList) returns sum of list of numbers
* <object> in myList returns true/false if item is in list

Iterating

* Use a for-loop to go one item at a time

for item in myList: # by Item/Object way

print(item)

for i in range(len(myList)): # by Index way

print(myList[i])

String’s .split() method

* myString.split() returns a list of the words in the string
* In the parentheses, you can specify what character you want to split the string on
  + myString.split(“,”) will split on commas (and removes the commas)

List Comprehension

* myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
* [<var> for <var> in <list>]
  + [i for i in myList]
  + [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
* [<var> for <var> in <list> if <var condition>]
  + [i for i in myList if i % 2]
  + [2, 4, 6, 8, 10]
* [<var expression> for <var> in <list>]
  + [i \* 3 for i in myList]
  + [3, 6, 9, 12, 15, 18, 21, 24, 27, 30]
* [<var expression> for <var> in <list> if <var condition>]
  + [i \* 5 for i in myList if i <= 4]
  + [5, 10, 15, 20]

Week 12

How to create a copy of a list

* myList = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
* newList = myList NO
* newList = [ i for i in myList] YES
* newList = [ ] + myList YES

Search a List

* Linear Search
  + [Code Link](https://bitbucket.org/cmano/cs1400-online-demo-code/src/master/Unit_6/Section_5/linear_search.py)
  + Will check all values in the list and return when it finds the search value
  + Worst case is it will have to check all n values
* Binary Search
  + [Code Link](https://bitbucket.org/cmano/cs1400-online-demo-code/src/master/Unit_6/Section_5/binary_search.py)
  + List must be sorted to use this
  + Will check one half of the list or sublist at a time, over and over.
  + Change the indexes searched each loop
  + Worst case is it will only have to check log(n) values (log base 2)

Sort a List - [Visualization Link](https://visualgo.net/bn/sorting) (Hit ‘esc’. The names of sorts listed at the top. Hit ‘Go’ to run)

* Bubble Sort
  + [Code Link](https://bitbucket.org/cmano/cs1400-online-demo-code/src/master/Unit_6/Section_6/bubble_sort.py)
  + The largest items get swapped to the back, one pair of indexes at a time
* Selection Sort
  + [Code Link](https://bitbucket.org/cmano/cs1400-online-demo-code/src/master/Unit_6/Section_6/selection_sort.py)
  + Find the lowest value in the unsorted portion of the list, and move it to the front
* Insertion Sort
  + [Code Link](https://bitbucket.org/cmano/cs1400-online-demo-code/src/master/Unit_6/Section_6/insertion_sort.py)
  + Keep the first portion of the list sorted, and add one value from the list at a time
  + Place the new value into the sorted portion where it belongs

Week 13

Multidimensional Lists

They’re just lists of lists

Can have any number of dimensions of lists -- but we’ll stick to 2D

How to create 2D list:

myList = []

for i in range(3): # will have 3 rows or 3 sublists

tempList = []

for j in range(4): # each row will have 4 items or 4 columns

tempList.append(randint(10, 99)

myList.append(tempList) # appending a list to a list

How to access data in 2D list:

One Item:

my2DList[rowIndex][columnIndex]

Every Item in a Loop by Index:

for rowIndex in range(len(my2DList)): # how many rows or sublists

for columnIndex in range(len(my2DList[rowIndex])):

# things in the current row

print(my2DList[rowIndex][columnIndex])

Every Item in a Loop by Item/Element/Object:

for rowList in my2DList:

for columnItem in rowList:

print(columnItem)

Ragged/Jagged Lists

Multidimensional lists where each sublist doesn’t have the same number of elements