Computer architecture = refers to the various parts of the computer and how they interact

Processor - computational heart of computer

Main memory - RAM

Disk - permanent storage

Input/output

Network - infrastructure that allows computers to communicate

Levels of hardware/software (lowest to highest)

1. Hardware abstraction layer (deepest we will ever go)

Handles inputs and outputs to the hardware

1. Basic Input/output System (BIOS)

Allows direct communication with peripheral devices

1. Operating System

Collection of programs that allow the user to interact with the hardware

1. Application programs

Office applications, games, etc.

CPU registers - fastest memory in computer (most expensive type of memory)

Works over short distances

Cache

1/10 the speed of registers - not as expensive

Main memory

Slower and cheaper than cache

All of these are volatile - info destroyed when turned off

Why we have external storage - never direct communication between internal and external

3 parts to program

1. Input - data from outside the program
2. Process
3. Output

Different types of programming languages

Machine language - lowest level that interacts with the computer

Efficient way to compute

No restrictions on language

Really tedious and need to rewrite the program if the hardware changes

Assembly Language

Takes code from programmer and translate it to machine language

Macro Assembly

Higher level of assembly language to reduce redundancy

One line of code can be multiple instructions to the computer

Higher level languages

Utilizes blocks of code to make it even easier to read and less tedious

If the hardware changes all we have to change is the complier

Within the higher level languages there are degrees on which how many details you control

IDLE is a REPL loop

Read, evaluate, print, loop

*Expression* - a combination of values and operations that creates a new value that we call a return value

  An expression is something that has a value

*Statement* - doesn't return a value but does perform some task

A statement does something

*Object*

Something that has an identity, some attributes, and zero or more names

Types inform us of

Attributes

Operations

Constructor is a special operation that is used to make a particular object of that type

Each type has an associated constructor

*Algorithm* - a method, a sequence of steps, that describes how to solve a problem or class of problems

3 issues for while loops

1. Initializing the control variable outside the loop
2. Control - initial value and the conditions are related and should be considered together
3. Modification - the variable gets modified so that it eventually becomes false

Sentinel loop - loop controlled by a sentinel (a guard) - a particular value to terminate the loop

The loop becomes defined on looping when it isn't set to a particular "guard" value

For vs while

While loops is more generla but the for loop is useful for moving through all the elements of a collection, think for first

*Id()* - returns the memory address to find that value

If variables have the same int value they will have the same id

This won't happen if they are float data types

When floating point numbers are stored in a computer it can never store exact numbers

This will lead to a slight rounding error

This is why .1 + .2 = 0.30000000004

You can have numbers in variables but the first character must be a alphabetical

\*\*\* Make sure you know the exact rules for naming conventions\*\*\*

Algorithm - a step by step procedure for accomplishing some task or solving some problem

Must be:

Complete - must work for any instance of the problem

Precise - must be spelled out in enough detail that it can be implemented

Unambiguous - at any point the next step is well defined

Terminating - it must eventually end

Correct - must get the right or best answer every time

Waterfall model

Requirement

Design

Implementation

Verification

Maintenance

Spiral model

Determine objectives

Identify and resolve risks

Development and test

Plan the next iteration

\*\*\* Practice using .format() make sure you know how to use it in a variety of situations\*\*\*

Earlier in the alphabet -> greater than letters later in the alphabet

Lower case -> greater than upper case letters

If blank compared to something -> something is greater

Compares character by character to find the first one that isn't equal and then compare them

a > z > A > Z >’ ‘

Stubs & drivers:

|  |
| --- |
| Refers to a method of stepwise development, with non-working function bodies gradually filled in as development proceeds. |

Functions

1. Divide and conquer
2. Abstrction
3. Reuse
4. Sharing
5. Security
6. Simplify code

Functions

Created with a function definition

Functions are called by invocation

Functions have their own namespace

Parameters are in the functions namespace

Function has to be defined before it is called

Functions name can't start with a number

Dir() to check the namespace (defined vars and functions)

Functions for lists

Len

Min

Max

Sum

Sorted - returns a new list with the same items in sorted order

Pop - removes a specific elemnt from list

l.extend

l.index

l.count

l.reverse

l.sort

L = L.sort() - destroys the list because .sort() doesn't return anything

Split() - breaks the string into a list of smaller strings

Join() - inverse of split

Deep copy vs shallow copy only matter with regards to when the copy has a mutable object

Shallow copies a change in the mutable object will be reflected in both copies

Deep copies a change in the mutable object will only be reflected in the copy you changed it for

Shallow copy - just copy the reference, if a change happens with one it will the effect the other, if you don't want the changes to effect the other copy you need a deep copy

Deep copy -

Import copy

C = Copy.deepcopy(A)

Score - the set of statements in which a particular variable can be referred to

Lifetime - duration between the time a variable is created and the time it is destroyed

Score refers to space, lifetime to time

Scope is the portions of your porgram where a variable exists

Lifetime is the period between creating and deletion of a variable or name

Scope Search order

Local

Enclosing - ex.` functions inside of functions

Global

Built-in - all the python functions and datatypes

Locals() - returns local namespace

Global() - " global "

If you use mutables as a default value when you call it as a named parameter it will change the parameter to the default instead of the mutable parameter

Annotation

Notes for what the parameters should be (not forced just a hint)

": " - for parameters

"-> " for returns

Ex.

Def function( a : int, b : str) -> int

|  |
| --- |
| def AppendValue( value, lst = [] ):       lst.append(value)       return lst    my\_list = AppendValue( 20 )  new\_list = AppendValue( 55 )    What is new\_list ? |
| |  |  | | --- | --- | | Selected Answer: | Incorrect    [55] | | Correct Answer: | Correct    [20, 55] | |

\_\_name\_\_

== "\_\_main\_\_"

Means it is being run from there, if not it is a module

When you add an if \_\_name\_\_ == "\_\_main\_\_" it will stop it from running the code of the main body when you import the code

Dictionaires

Methods

.clear() - empties the dictionary

a.update(dict1) each key in a is updated with the key-value pair from dict1

.copy() - make a shallow copy

.items() - all key value pairs

.keys() - all the keys

.values() - all the values

Functions

Len()

Min()

Max()

Sorted()

Returns sorted list of the keys

Keys()

Values()

Items()

Three ways to update a key

Use try except

Pop off the value and add a new one in that place

Get method

2 arguments - 1 is optional

Key, default value

Gets the value assigned to that key if it isn't there it assigns the default value

Sets

Unordered, mutable collection of data

Similar to sets you learned in math class

Duplicates are not supported (item is in the set or not but no duplicates)

Python considers 2 and 2.0 int vs float the same in sets (it will only keep one if the number value is the exact same)

Why use sets?

Get unique elements

Find shared elements between t2 sets

Combine 2 lists, while removing duplicate elements

Showing the difference between sets

Functions

.add

.remove

.pop

.clear()

.update() - merge sets

.intersection

.union

.difference - makes a new set of the elements of the first set that are not in the second

.symmetric\_difference() - everything in the unique but not the intersection

.issuperset

.issubset

.remove() - raises an error if the item isn't there

.discard() - doesn't raise an error if the item isn't there

.copy

You can do set and dictionary comprehensions that operate similar to list comprehensions

S = {k for k in range(10)}

D = {k:2\*k for k in range(10)}

Frozen sets

When you make a set immutable

Operates similar to regular set with the modification of add and removing objects from it

Good if you want to use a set as a dictionary key

Characteristics of OOP

PIE+M

P - polymorphism - objects act different by the class they are

I - Inheritance - ability for code to build off itself

E - Encapsulation - hiding how it works

When you call functions you don't need to know how it operates to use it

Modularity - an object that can be consumed by any number of applications

Convention for class naming is camel case

Best way to add attributes and methods is when you initially define the class

\*\*\*How do we add attributes after initialization?\*\*\*

3 types of errors

Syntax

Runtime - program runs but has an error while executing

Design Errors - program runs but not correctly

Testing for

Correctness

Completeness - are all the potential cases covered

Security

Interface - is it usable?

Load - how does it perform under stress

Resources - does the app handle its resources correctly (closes files, releases memory, etc.)

Responsiveness - does the program respond in a timely fashion

TDD - Test Driven Development

If you write the tests first you will write more robust code

2 types of testing we will talk about

DocTesting - use the doc stringwith parameters passed to declare any tests

UnitTesting - can be included in an external module to test other modules

 Testing is a form of documentation

Unit tests

When testing functions in class it must begin with test

The class must inherit from 'unittest.TestCase'

Must import "unittest" first

Assert - if you put assert before a boolean statement it will raise an assertion error when the expression evaluates to false

Most python users don't like assert

Understand unbound method calls

Bound method calls are "attached" to a specific object

When you call an object that is tied to a specific object it is unbounded

Ex.

l.sort() vs list.sort(l)

Important when using parent and child classes

When using \_\_init\_\_ for the child you can make an unbound method call for the parents \_\_init\_\_ so any changes are reflected in the child class

Multiple exceptions handled the same way

Try:

Except (indexError, typeError):

Else: (if it works correctly)

Finally: (executed before leaving try block)

Be able to subclass a class, subclass and error, create custom errors and raising errors

Know about the os module and its functions

Change directory

Walk

Etc.

Split extension

Join

Exceptions

How to raise and subclass exceptions

Operator overloading

\_\_add\_\_, \_\_radd\_\_,

Etc.

Know what each one of these does for each operator

Testing

Assert statement

If it is true it will continue to the rest of the code

If it isn't it will raise an assert exception

Be able to doctest and unittest

He may write unittests and be able to say if it passes or fails

\*\*\* Play along with doc and unit testing until you are comfortable with it \*\*\*

Reviewing Test Study Guides

Literals

Namespace

Expression vs statement

Chained relational operators

Chained assignments

Check for tolerance

Ord(), chr(), id()

Function invocation

Scope LEGB

PIE and M

Class contains attributes and methods

Standard methods for classes (\_\_init\_\_, \_\_str\_\_)

Introspection – type(), isinstance()

Kinds of errors

Syntax, runtime, design

Static vs dynamic testing

Post vs pre testing

Levels of testing

Unit testing – a unit at its lowest level

Integration testing – do units work together?

System Testing – does whole system work together

What to test

CCSILRR

Assert statement (assertions)

.read(n\_bytes), write(s), writelines(lst)

File position

Tell()

Seek(offset, from\_where)

With statement

Os module

Os.getcwd()

Os.chdir(str)

Os.listdir(str)

Os.walk(str)

Os.path module

Os.paht.isfile(str)

Os.path.isdir(str)

Os.path.exists(str)

Exceptions

Philosophy

Lbyl – look before you leap, check everything

EAFP – easier to ask forgiveness than permission

Try else except finally

Raise an exception

Subclassing and making own exceptions