Python Notes

Python

* No need for a main loop
* Python doesn’t need to initialize variables?? Age=20; print(age);
* No { or ; in python

* + If x < 2:
    - Print ‘less than 2’
  + Else:
    - Print “not less than 2”
* Indentation is used to break up code not the typical brackets
* Print “hey” + str(x)
* Functions are just as simple, use def to initialize the function not the type
  + Def foo(x):
    - Return x \* 5
* To inherit classes you don’t write (class foo2 extends foo) you write class foo2(foo):
* Constructors no longer the name of the class, they are called def \_\_init\_\_(x, y)
* This.x equals self.x
* When getting data from the main stream it reads a string not as integers
* In math, / is a float and // is an int
* When using arrays name[-1] is the last name on the list and so on, names[0:3] prints the first three values
* Tuples are immutable lists made by x = (1,2,3). Can never change

Functions

* Input(“”) – gets input from main stream
* Print(“” + x) – must be of same type to print
* Str() - convert to string, int() – to integer, float() – to float, bool() - to Boolean
* / - float division, // - int division, x\*\*3 - to the power of
* And, or, not – all are text not symbols
* If, elif, while
* Names = [“John”, ”Bob”, ”Sam”] – lists are basically arrays
* Names.append(6) – adds 6 to the end
* Names.insert(0,6) – inserts 6 at index 0
* Names.clear() – clears list
* For items in names: - the for loops to run through an array
* For number in numbers: - gives number for loop creating a variable number
* Range(5,10) - generates the sequence of numbers between the values

Lists

* Written as myList = [1,2,3,4,5,…]
* X=”hello”, x.upper() – prints HELLO,
* find(‘a’) – prints where the first value is located in a string, or -1 for null
* index(‘a’)- also returns first index value
* if “” in list:
* replace(‘a’ , ‘b’) – changes all values of a to b
* Append(“”)-puts at end
* Pop()- returns value popped from top
* remove(“”)-specific item
* clear()
* reverse()- also done with b=a[::-1]
* sort()
* list = [0] \* 5 – makes a list of 5 zeros
* x=list[1:5] - prints new list with the first 5 elements of the original
* b = [i\*i for i in myList] - create new list of a’s squares
* “hello” in X – prints 1 or 0 for if the string hello is in x
* Setting lists equal to each other aliases both lists, linking them
* List comprehension – (names: list[str] = [p for p in people if len(p) >7]) creates a list of names for every name in people that are longer than 7 characters

Tuples

* Written as myTuple = (“he”, 5, “hello”) or myTuple = “he”, 5, “hello”
* Cannot be changed
* Tuple([“max”,28,”hey]) – makes a tuple from a list
* Count(‘p’)-counts the number of p’s
* x, y, z = myTuple – assigns values
* x, \*y, z = myTuple – saves fist and last to x, z. and the rest to y in a list []

Dictionary/map

* Written as myMap = {“key”: “value”,” key”: “value”,…}
* Or we can myMap = dict(key = “value”, key =”value”,…}
* They will alias and be linked
* Work with tuples and not lists
* Value = myMap[“key”] lets you search for items
* myMap[“key] = “value” – add a pair to the map
* del myMap[“key”] and myMap.pop(“key”) – delete values
* .popitem() – removes the last inserted item
* Is “key” in myMap: or try: except: - to check if an item is inside
* For keys in myMap: runs through each pair

Sets

* Created by mySet = {1,2,3}
* Cannot have duplicates/ unordered
* .add(1) – adds to randoms spot
* .discard(1) – deletes
* .clear() – resets the set
* .pop()-pulls out and returns a random value
* Set1.union(set2) – combines sets without duplicating values
* Set1.intersection(set2) –

Strings

* Strings are Immutable
* b=a[::-1] reverses string
* a = “hey” + “bob” concatenates strings
* for \_\_ in \_\_ works to looks for specific letters
* .upper / .lower / .startswith / .endswith / .find(“”) / .count(“”) / .replace(“”, “”)
* .split() – breaks string into list(by spaces unless its .split(“,”) splits by commas
* ‘ ‘.join() concatenates a list into a string with spaces

Collections

* From collections import \_\_\_\_
  + Counter
  + numedtuple
  + orderedDict
  + deque – double sided queue

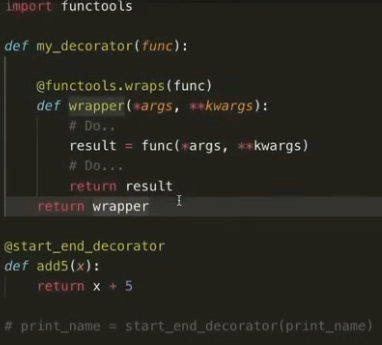
lambda arguments

* initialized by name = lambda var: expression
* one line functions(add10 = lambda x: x+10
* can be recalled with any argument by add10(y)

Logging

* Logging.debug(‘’)
* Used instead of print statements
* Logging has 5 levels, debug, info, warning, error, and critical
* .basicconfig(level = ‘’, format=’’, datefmt=’%m/%d/%Y %H:%M:%
  + S’)- lets you change the setup
* Name.setlevel(logging.warning) – set the new level to read from
* In a new file write loger=logging.getLogger(\_\_name\_\_) to set up the logger form a different file and then import the other file into main
* Stream\_h = logging.StreamHandler() – logs to the default iostream
* file\_h = logging .fileHandler(“file.log”) – logs to a file
* a file called logging.conf is used in order to set up many loggers and specify their formatting and useage, look more into the format of that file, then import logging.config to use in each file.

Decorators

* function decorator - extends functionality of function without changing it
* *using the wrapper you must save the result of the function and return it out of the decorator*
* 
* You can put things before and after the function without changing it
* Using def wrapper (#arrgs, ##kwargs): when defining lets you use as many arguments and keyword arguments as you need in the function

Generator

* Def mygenerator(): {yield 1 yeild 2 yeild 3} creates an object with three separate returns
* Can be used to save memory by easily letting you make arrays in a function without creating the function
* For I in myGenorator – prints all the yield statements from the generator
* Value = next(mygenorator) – runs through the function until the first yield and returns it, remembering where it ended. Next time its called it will start at the yield and finish the code before restarting to get to the yield.
* any function that takes an iterable as a input like sum() or sorted() will sum the each yield together and work with that
* genorator comprehension – names = (p for p in people if len(p) >7), similar to list comprehension but with () instead of []

Threading vs Multiprocessing

Process

* instance of a program (such as c file)
* have separate memory space
* started independently and are interruptible
* can be run simultaneously on different CPUs
* larger

game

* Mario style
* Par core
* Zelda style
* Have the screen open and close new windows when you go into another room

Multithreading

* entity of a process (functions)
* lightweight
* Great for I/O tasks
* can only run one thread at a time
* from threading import Thread

A screen shot of a computer code

AI-generated content may be incorrect.