lower\_snake\_case for variables, modules, function names

UPPER\_CASE for constants (whose values will not change)

UpperCamelCase only for classes (covered in lesson 11)

\n

\t

End=’ ‘

The built-in function id() will return the id of an object,

// integer division

\*\* exponent

%remainder

%% percentage sign

Function is named as a verb: divide the code into smaller modules

Pass: stub it out

SyntaxError

DocStrings provide a description of use and functionality of your code for the intended audience (other developers). it helps the developer to understand the code, its purpose and design

while

**Break**: terminate the loop; the flow of control will jump to the statement directly after the while loop.

**Exit**: terminate the program (note: this is rarely used).

**Return**: exit from a function.

**Continue**: skip the rest of the current iteration of a while loop

\”name\”

+concatenate string

\*repeats string

Zero padding: %03d

string

**%**: “%s purchase %s from store for %.2f CAD”%(name,fruit,price )

**f-string**: f”{Sama} purchased {apple} from store”

**f-string+%:** f”{Sama} purchased {apple} from store for %.2f CAD”% price

**.format**:”{} purchased {} from store”.format(“sama”,”apple”)

**csv:** “Susan,Lee,18,34.58”

**JSON=[{},{},{}]**

String characters do not change **immutable**

**Index[-1]** identifies the last character

**Out of bound index is ok for string**

Split: splits the string into a list based on a separator message**.split(,)**----🡪 list

Message**=” “.join**(token) list to str

**replace(old, new)** -- Returns a copy of the string with all occurrences of the substring old

replaced by the string new. The old and new arguments may be string variables or string

literals.

**replace(old, new, count)** -- Same as above, except only replaces the first count

occurrences of old.

**find(x)** -- Returns the position of the first occurrence of item x in the string, otherwise

returns -1, if x was not in the string. x may be a string variable or string literal.

**find(x, start)** -- Same as find(x), but begins the search at position start

**find(x, start, end)** -- Same as find(x, start), but stops the search at position end

**rfind(x)** -- Same as find(x) but searches the string in reverse, returning the last occurrence

in the string.

**count(x)** -- Returns the number of times x occurs in the string.

**isalnum()** -- Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.

**isdigit()** -- Returns True if all characters are the numbers 0-9.

**islower()** -- Returns True if all characters are lowercase letters.

**isupper()** -- Returns True if all cased characters are uppercase letters.

**isspace()** -- Returns True if all characters are whitespace.

**startswith(x)** -- Returns True if the string starts with x.

**endswith(x)** -- Returns True if the string ends with x.

**capitalize()** -- Returns a copy of the string with the first character

capitalized and the rest lowercased.

**lower()** -- Returns a copy of the string with all characters lowercased.

**upper()** -- Returns a copy of the string with all characters uppercased.

**strip()** -- Returns a copy of the string with leading and trailing

whitespace removed.

**title()** -- Returns a copy of the string as a title, with first letters of words capitalized.

**is operator** checks memory addresses

tuple and list

**tuple:** immutable ()

tuple\_name.count()

**list:** mutable []

**letters.pop()** # removes the last element

**letters.remove**("y") # removes the first instance of y

**letters.append("d")** # appended to the end of the list

**cities.extend**([“jhg”,”hgg”])

all\_letters = **letters + more\_letters** # joins together (“concatenates”)

**countries.index(**"england")

province for province in provinces if "e" in province

**del cities[1**]

**cities.reverse()**

for

**range(start, stop, step)** just integers, does not include end

**break** can be used to terminate any loop prematurely

**continue** is used to immediately stop the rest of the current iteration and continue to the next iteration

**enumerate**: for index, city in enumerate(cities)

set

no duplicate, not stored in a particular order, can’t change items

my\_set = {“hello”, 1, 2} or my\_set = set([“hello”, 1, 2])

Add one item using add() my\_set.add(“bye”)

Add multiple items using the update my\_set.update([4,8,9])

Remove: my\_set.remove(“1”) if 1 does not exists causes error

: my\_set.discard(“1”) if 1 does not exists no error

The clear() method empties the set

The del keyword will delete the set completely: del my\_set

union(): new\_set = my\_set.union(that\_set)

Dictionaries

key-value pairs

dictionary\_name = { “key\_name” : “value”, “key\_2\_name” : “value 2”, “key\_3\_name”: “value 3”}

dictionary\_name.get(“key\_2\_name”, **“not found”**) or dictionary\_name[“key\_2\_name”]

add or edit key: dictionary\_name[“key 4”]=”value 4”

delete a key: del dictionary\_name[‘key\_3\_name]

Remove all items from a dictionary using the **clear()** method

Merge dictionaries together using the **update()** method

Remove and return the key value from the dictionary using the **pop(‘key\_3\_name’,’not found’)** method

dictionary\_name.keys()

dictionary\_name.values()

for key, value in dictionary\_name.items()

file

with open(‘books.txt’, ‘r’) as file:

note = file.read()

print(note)

or

file = open(‘books.txt’, ‘r’)

note = file.read() **retuns a string**

print(note)

file.close()

print(repr(note)): it prints all unseen characters like enter as well ‘python crash\neffective python’

print(file.readlines()): **returns a list** of all the file lines including the “\n” at the end of each line

[‘python crash\n, ’effective python’]

If the file size is huge, loop can be used to read the file one line at a time:

For line in file:

Print(line.strip())

‘r’ read

‘w’ write creates a new file to write or overwrite if the file exists

‘a’ append to the end of the existing file

Debug

1. watch window: see what the program is storing in variables

2. stack trace: see where the program is coming from and going to

3. step: one at a time, execute (and/or skip) individual

instructions

Regex

Re.findall

Re.match find the first occurrence

Re.search

Re.split(“ ”, string1)

Re.finditer returns an iterator yielding a match object of all the occurrences

Re.search("a", string,re.IGNORECASE)