Wk4, Lec7

test question: how many times will this be printed?

def factorial(n):

answer=1

if n>1:

for k in range(1,n=1):

answer=answer\*k

print(‘ha ha ha’)

return answer

# print is not in for loop, so it will only print once.

Wk6, Lec11

method may or may not mutate

functions do not mutate

* do not have any implicit parameters
* functions are just value-returning things

diff btw sort and sorted is method and functions

Wk6, Lec12

implicit/explicit parameter = aka arguments

* implicit is within ()
* explicit is outside ()

l.sort = ‘sort’ is a method, not function

method is part of a class (blueprint for the data type)

* has instance
* also method

1. expression = just something that evaluates to something (takes a value)
2. asdf
3. asdf

you can’t change strings; you can reassign strings to different variables, but you can’t actually change strings.

Dictionary: (def.) a data structure

set can’t have repeat values = \*key feature of a set

* dictionaries are pairs of things, sets are not

Wk7, Lec14

* “shallow copy”
* always go through the following steps:
  + how many objects there are: 4: [1,2,3, and list holding it]
  + how many object references there are: 4: L1 referring to list, 3 referring to 3 integers (L1[0], L1[1], L1[2])
  + any new objects on the next line? no
  + four new object references: L2[0], L2[1], L2[2]
* difference between using the “assignment operator” and “mutating” and object, e.g. append
  + one is using variable assignment, the other is using mutation
  + strings you can’t change
* “elemental data type” – “a little”
* re-assign vs. copy:
  + a={1, 2, 3}
  + b=a
  + a.append(4)
  + b is now {1, 2, 3, 4}
  + vs:
    - a={1, 2, 3}
    - b=a{ }
    - a.append(4)
    - b still equals {1, 2, 3}
* tuples are like “immutable” lists
* \*\*\* there will be a question on the quiz about tuples and how they differ from lists
* aliases = just another object reference that just so happens to be pointing to the same object
* “scope” of a variable – where that variable exists; what part of the code the variable is in
* \*\*\* the formal parameters are the names with give to parameters in the function definition
* what is x? don’t tell me x is a list; tell me x is a variable pointing to a list.
* \*\*\* **keys** must be of a type that is immutable; there should also be only one of each in a dictionary; must be unique and cannot change (the value can)
* Problem 3: change to a set, change it back to a list and now it’s missing what you wanted to leave out.

Wk8, Lec15

\*\*\* (4pts) what does it mean when you import [function] “as” [blank]

* u can pull up

what would range[10] return?

0-9

range returns integers

what would range[2:10] return?

what would range[2, 10, 2] return?

lind space == I like this one. don’t forget it.

\*\*\* dec.5 will test on “important Numpy methods”

trust testing this concept again: “order of precedence”: “please excuse my dear aunt sally”

def f(a,b):

return 2\*a+b

a=2

b=4

f(b,a) (this is not the same as the a,b in the function, i.e. it is outside the function)

= 10

**“positional parameters” because they get their values positionally**

str.split(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most maxsplit+1 elements). If *maxsplit* is not specified or -1, then there is no limit on the number of splits (all possible splits are made).

If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, '1,,2'.split(',') returns ['1', '', '2']). The *sep* argument may consist of multiple characters (for example, '1<>2<>3'.split('<>') returns ['1', '2', '3']). Splitting an empty string with a specified separator returns [''].

For example:

>>>

>>> '1,2,3'.split(',')

['1', '2', '3']

>>> '1,2,3'.split(',', maxsplit=1)

['1', '2,3']

>>> '1,2,,3,'.split(',')

['1', '2', '', '3', '']

If *sep* is not specified or is None, a different splitting algorithm is applied: runs of consecutive whitespace are regarded as a single separator, and the result will contain no empty strings at the start or end if the string has leading or trailing whitespace. Consequently, splitting an empty string or a string consisting of just whitespace with a None separator returns [].

For example:

>>>

>>> '1 2 3'.split()

['1', '2', '3']

>>> '1 2 3'.split(maxsplit=1)

['1', '2 3']

>>> ' 1 2 3 '.split()

['1', '2', '3']

str.rsplit(*sep=None*, *maxsplit=-1*)

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or None, any whitespace string is a separator. Except for splitting from the right, [rsplit()](https://docs.python.org/3/library/stdtypes.html#str.rsplit) behaves like [split()](https://docs.python.org/3/library/stdtypes.html#str.split) which is described in detail below.

str.isdigit()

Return true if all characters in the string are digits and there is at least one character, false otherwise. Digits include decimal characters and digits that need special handling, such as the compatibility superscript digits. This covers digits which cannot be used to form numbers in base 10, like the Kharosthi numbers. Formally, a digit is a character that has the property value Numeric\_Type=Digit or Numeric\_Type=Decimal.

str.strip([*chars*])

Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or None, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

>>>

>>> ' spacious '.strip()

'spacious'

>>> 'www.example.com'.strip('cmowz.')

'example'

The outermost leading and trailing *chars* argument values are stripped from the string. Characters are removed from the leading end until reaching a string character that is not contained in the set of characters in *chars*. A similar action takes place on the trailing end. For example:

>>>

>>> comment\_string = '#....... Section 3.2.1 Issue #32 .......'

>>> comment\_string.strip('.#! ')

'Section 3.2.1 Issue #32'

str.rstrip([*chars*])

Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or None, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

>>>

>>> ' spacious '.rstrip()

' spacious'

>>> 'mississippi'.rstrip('ipz')

'mississ'

str.lstrip([*chars*])

Return a copy of the string with leading characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or None, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix; rather, all combinations of its values are stripped:

>>>

>>> ' spacious '.lstrip()

'spacious '

>>> 'www.example.com'.lstrip('cmowz.')

'example.com'

str.replace(*old*, *new*[, *count*])

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.