

recursion practice: <https://www.w3resource.com/python-exercises/data-structures-and-algorithms/>

python-recursion.php

solving problems:

always check that something exists in the parameters of what is passed to the function

inter.sort(key = lambda x: x[0])

try to use sort - especially if some type of order is necessary

Python

>>>

```
>>> ids = ['id1', 'id2', 'id30', 'id3', 'id22', 'id100']
>>> print(sorted(ids)) # Lexicographic sort
['id1', 'id100', 'id2', 'id22', 'id3', 'id30']
>>> sorted_ids = sorted(ids, key=lambda x: int(x[2:])) # Integer sort
>>> print(sorted_ids)
['id1', 'id2', 'id3', 'id22', 'id30', 'id100']
```

In [69]: prime_numbers = [2, 3, 5, 7, 11, 13, 17]

In [70]: removed_element = prime_numbers.pop(2)

In [71]: removed_element

Out[71]: 5

In [72]: prime_numbers

Out[72]: [2, 3, 7, 11, 13, 17]

In [73]: removed_element_2 = prime_numbers.pop(-2)

In [74]: removed_element_2

Out[74]: 13

Operation	<u>Linked List</u>	<u>Python's list</u>
Insert at head	O(1)	O(n)
Insert at middle	O(n), O(1) given a reference	O(n)
Remove from head	O(1)	O(n)
Remove from middle	O(n), O(1) given a reference	O(n)
Find k-th element	O(k)	O(1)
Search unsorted	O(n)	O(n)
Search sorted	O(n)	O(log n)

python comparisons is type sensitive

reversed(array)

```
half_n, remainder = divmod(n, 2) # n // 2, n % 2
```

divide + mod, at the same time

update existing

dict: dict.update({new: values})

list: list.append([new, stuff])

set: set.add({new, stuff})

```
1 person = {}
2
3 # Using get() results in None
4 print('Salary: ', person.get('salary'))
5
6 # Using get - define default value
7 print('Salary: ', person.get('salary', 0.0))
8
9 # Using [] results in KeyError
10 print(person['salary']) 56
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
ayeletwasserman@Ayelets-MacBook-Pro test_prep % /usr/local/bin/p
Salary:  None
Salary:  0.0
Traceback (most recent call last):
  File "/Users/ayeletwasserman/Documents/CS/intro/test_prep/dict
    print(person['salary'])
KeyError: 'salary'
ayeletwasserman@Ayelets-MacBook-Pro test_prep %
```

```
In [22]: txt = "banana,,,.ssqqqww....."
...: x = txt.rstrip(",.qsw")
...: print(x)
banana
```

from **collections** import Counter

list.remove(item_to_remove) - removes only the first occurrence. error if trying to remove value that

does not exist

decorators - without parenthesis

from **functools** import wraps (@wraps(func) - get the original info of a wrapped func)

@property:

```
class Person:
    def __init__(self, name, age):
        self._name = name
        self._age = age

    # Define a "name" getter
    @property
    def name(self):
        return self._name

    # Define a "name" setter
    @name.setter
    def name(self, value):
        self._name = value

# Create a new Person object
person = Person("Alice", 30)

# Access and set the "name" attribute using the getter and setter
print(person.name) # "Alice"
person.name = "Bob"
print(person.name) # "Bob"
```

```
class Person:  
    def __init__(self, name, age):  
        self._name = name  
        self._age = age  
  
        # define a "name" getter  
        @property  
        def name(self):  
            return self._name  
  
        # define a "name" deleter  
        @name.deleter  
        def name(self):  
            del self._name  
  
    # create a new Person object  
    person = Person("Alice", 30)  
  
    # access, set, and delete the "name" attribute  
    # using the getter, setter, and deleter  
    print(person.name) # "Alice"  
  
    del person.name  
    print(person.name) # AttributeError
```

python arithmetic = leaves floating point stuff

```
>>> 1-1/3 == 2/3  
False
```

```
>>> 1-1/3  
0.6666666666666667  
>>> 2/3  
0.6666666666666666
```

```
>>> sys.getsizeof(1/3)  
24  
>>> sys.getsizeof(2.0**500)  
24
```

```
type(1 + 0.5) == type(float)
```

```
int(5.9) == 5, int rounds floats down
```

```
float(int) == int.0
```

String Methods

Python has a set of built-in methods that you can use on strings.
For example:

<u>upper()</u>	Converts a string into upper case
<u>split()</u>	Splits the string at the specified separator, and returns a list
<u>replace()</u>	Returns a string where a specified value is replaced with a specified value
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string
<u>startswith()</u>	Returns true if the string starts with the specified value
<u>strip()</u>	Returns a trimmed version of the string

```
string.split('what to split with')
```

```
'hello'.startswith('o') == False
```

```
'hello'.startswith('hel') == True
```

```
[4,5,3,5,3].count(3) >>> 2
```

```
string.replace(oldvalue, newvalue, count)
```

range does not include the last value

```
type(range(2))  
range
```

```
list1 = list(range(5))  
print(list1)  
list1[0:1] = [1, 2, 3]  
print(list1)
```

[0, 1, 2, 3, 4]

[1, 2, 3, 1, 2, 3, 4]

```
list1 = list(range(5))  
print(list1)  
list1[0] = [1, 2, 3]  
print(list1)
```

[0, 1, 2, 3, 4]

[[1, 2, 3], 1, 2, 3, 4]

break - terminates the current loop and continues with the rest of the program

continue - terminates only the curr iteration

```
lst3 = lst[:end] # items from the beginning through end-1
```

```
lst4 = lst[start:end:step] # items from start through end-1  
                           by step
```

```
lst5 = lst[::-step] # from the beginning to the end by step
```

list.append(x) – Add **one element** to the end of the list

- equivalent to `a[len(a):] = [x]`

append() takes exactly one argument!

`list.insert(index, obj)` - Inserts object obj into list at offset index

list.extend(seq) - Appends the contents of seq to list

list.sort([func]) - Sorts objects of list, use func to compare, if given

`ord('A')` = 65, `ord('a')` = 97. opposite: `chr()` to convert back to character form

```

x = [[1, 1, 1]] * 3
print(x)
x[0][0] = 0
print(x)

```

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

```
[[0, 1, 1], [0, 1, 1], [0, 1, 1]]
```

```

x = [1, 2, 3]
y = copy.copy(x)
z = list(x)

print(y is x)    #False
print(z is x)    #False

```

```

x = "hi"
y = "bye"

z = x + y          # z is a new string!

w = z[2:4]         # slicing creates a copy
# and doesn't change
# the original string...

x[0] = 3           #ERROR!
x.append("a")      #ERROR!

```

ranges are immutable

Immutable types are often reused

(and sometimes not)

```

x = "hello"
y = "he" + "llo"

print(x is y) True

x = (0, 1, 2)
y = (0,) + (1, 2)

print(x is y) True

```

```

x = range(10)
y = range(10)

print(x is y) False
print(x == y) True

```

```

x1 = [[1] * 3] * 3
x2 = [[1] * 3 for i in range(3)]
x3 = [[1 for i in range(3)] for i in range(3)]
x1[0][0] = 9
x2[0][0] = 9
x3[0][0] = 9
x1
[[9, 1, 1], [9, 1, 1], [9, 1, 1]]
x2
[[9, 1, 1], [1, 1, 1], [1, 1, 1]]
x3
[[9, 1, 1], [1, 1, 1], [1, 1, 1]]

```

`==` does deep comparison.

Files:

```
f = open(filename,'a')
```

```
n = f.write(s)
```

- `f.writelines(seq)`

- `f.close()`

- `f.read(n)`

```
f.readline()
```

```
f.readlines()
```

```
with open(file_path, 'w') as our_open_file:  
    # we can work here with the open  
    # file and be sure it is properly  
    # closed in every scenario
```

IF `__NAME__ == "__MAIN__"`: `__name__` will have the value "`__main__`" only if the current

module is first to execute.

Quick Summary - Aliasing

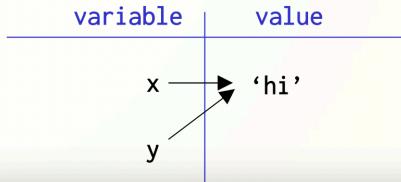
Copying with the `=` sign:

- Using the assignment operator (`=`) **always** copies just the identity of the variable on the right to the variable on the left (no matter what value the right variable has) – so the value itself is **not** being copied.

copying a reference

- When we **modify** the variable's value, we making a **new identity** for that variable.

```
>>> x = "hi"  
>>> y = x
```



is - tests for identity

== compares the values themselves

```
>>> def init_student_num():
    global num_student
    num_student = 0
```

```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[y for x in list_of_lists for y in x]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

enumerate(seq, **[start_index]**)