PYTHON OPERATORS

Arithmetic operators						
+	-	*	**	/	//	%
sum	Minus	Mult	Power	Div	Floor div	Remain
Relational operators						
>	>=	<	<=	==	!=	=
						Assigning
Boolean operators						
or	and	not				

ORDER OF GROWTH

Time complexity:
Recursive: usually O(n), total operations
Iteration: usually O(n), total steps
Space complexity:
Recursive: usually O(n), total pending operations
Iteration: usually O(1), total variables used

LIST

List is mutable.
lst = [1, 2, 3]
Changing an element in a list:
lst[0] = 5
return lst → [5, 2, 3]
Deleting an element in a list:
del lst[0]
return lst → [2, 3]
Sorting a list:
lst = [('a', 2), ('c', 4), ('b', 6)]
lst.sort(key, reverse)
key = can be x:x[0], means sorted by 'abc'
reverse = True → descending

List operations

.append(n)	Append an element/list to a list *n can be a list too lst = [1,2,3] lst.append([4, 5, 6]) return lst → [1, 2, 3, [4, 5, 6]]
.extend(n)	Append a list to a list lst = [1,2,3] lst.extend([4, 5, 6]) return lst → [1, 2, 3, 4, 5, 6]
.copy()	Returns a shallow copy of the list
.insert(n, p)	Inserts n to the position p in the list
.pop(p)	Removes the element in position p and returns that element *if it's lst.pop(), the last element will be removed
.remove(n)	Removes the first occurrence of n in the list, reports error if n is not found
.clear()	Removes all element in the list
.reverse()	Reverse the whole list

TUPLE

Operations	Returns
foo	the tuple foo
foo[0]	the first element
foo[-1]	the last element
foo[1:]	rest of foo without 1st element
foo[a:b]	a tuple consisting elements from index a to b (no b)
foo[a:b:c]	a tuple consisting elements from index a to b (no b), in steps of c
len(foo)	the number of elements in foo

DICTIONARY

A dictionary is a sequence of key-value pairs, keys must be unique and immutable (ie. Can be tuple but no list!).
The order of the dictionary is not important.
d = {'a': 1, 'b': 2}
To update an existing entry:
d['a'] = 5 → d = {'a': 5, 'b': 2}
To add an entry:
d['c'] = 0 → d = {'a': 5, 'b': 2, 'c': 0}
To retrieve all keys into a list:
list(d.keys()) → ['a', 'b', 'c']
To retrieve all values into a list:
list(d.values ()) → [5, 2, 0]
To clear the entries:
d.clear() → {}

OOP

Refer to a method defines previously
we have to add a () behind. Eg. self.get_minutes()
Refer back to a property defined previously
we don't have to add a () behind. Eg. self.minutes
Inheritance
super().__init__(name, age)
*where name and age is inherited from the parent class
Check the item's class
isinstance(item, class)
What if a property in a class is arbitrary?

```
class Animal(LivingThing):
    def __init__(self, name, health, food_value, threshold=None):
        if threshold == None:
            threshold = random.randint(0,4)
        super().__init__(name, health, threshold)
        self.food_value = food_value
```

EXCEPTION HANDLING

SyntaxError: invalid syntax	NameError: name 'spam' is not defined
TypeError: can only concatenate str (not "int") to str	ValueError: invalid literal for int() with base 10: 'one'
ZeroDivisionError: division by zero	

EXCEPTION HANDING FORMAT

```
try:
    # statements
except Error1:
    # handle error 1
except Error2:
    # handle error 2
except: # wildcard
    # handle whatever error
else:
    # do this
    # if no error raised
finally:
    # always do this block
```

TAKE NOTE

1. Never return a list modification, eg. return lst.append(1), instead, use newlst = lst.append(1), return newlst
2. When a tuple only has one element, never forget to include ',' behind.
3. Never forget the 'return'
4. Never forget the ':'
5. Never operate on incompatible types. Eg. 5 + '1'
6. Check indentation in a loop to avoid infinite loops
7. If using while loop, remember to update the 'i'
8. Never forget the * during multiplication
9. For OOP, never forget to put (object) behind the class
10. For OOP, never forget to include 'self'.
11. Never reference a global variable inside a local frame.

OTHERS

1. can use 'in' or 'not in' to check whether an element is in a **list, tuple, dictionary**

*for dictionary, it is used to check the key, not the value

2. Anonymous functions → lambda <input>: <output>
3. length of strings can be compared with '<' and '>'
4. Counting leaves

```
def count_leaves(tree):
    if tree == ():
        return 0
    elif is_leaf(tree):
        return 1
    else:
        return count_leaves(tree[0]) + count_leaves(tree[1:])
```

```
def is_leaf(tree):
    return type(tree) != tuple
```

5. Flattening tree

```
def enumerate_tree(tree):
    if tree == ():
        return ()
    elif is_leaf(tree):
        return (tree,)
    else:
        return enumerate_tree(tree[0]) +
               enumerate_tree(tree[1:])
```

Example that will give error:

```
x = 10
def f(y):
```

SELECTION SORT AND MERGE SORT

```
def selection_sort(list):
    sorted = [] # sorted list
    while list: # i.e. while list is not empty
        smallest = list[0]
        for element in list:
            if element < smallest:
                smallest = element
        list.remove(smallest)
        sorted.append(smallest)
    return sorted
```

```
def merge_sort(lst):
    if len(lst) < 2:  # Base case
        return lst
    mid = len(lst) // 2
    left =  merge_sort(lst[:mid]) # sort left
    right = merge_sort(lst[mid:]) # sort right
    return merge(left, right)
```

```
def merge(left, right):
    result = []
    while left and right:
        if left[0] < right[0]:
            result.append(left.pop(0))
        else:
            result.append(right.pop(0))
    result.extend(left)  # either left
    result.extend(right) # or right is []
    return result
```

GENERAL BUILT-IN FUNCTIONS

min(), max()	Find the min/max from **list, tuple, dictionary** *applicable to alphabets too, the earlier alphabet like 'c' is smaller than the later alphabet like 'f' *if used at dictionary, it **returns the key** of the max/min value
len()	Find the length of a **list, tuple, dictionary** *in dictionary, 1 key-value pair = 1 unit
sum()	Find the total sum of the elements in a **list, tuple, dictionary** *all elements must be int/float *for dictionary, it means the sum of the keys, not the values, provided that all the keys are int/float, regardless of the values type
.upper() .lower()	Change a string to all upper case/ lower case eg. string = 'AbcDe' string.upper() → 'ABCDE'
.isupper() .islower()	Check whether the characters in a string is upper or lower *returns True only when all the characters in the string are upper case/lower case, otherwise False
round(n,d)	Round n to d decimal places