In [1]:
```
['rahul', 'vidhatri', 'sanika', 'vidhatri']
[10, 20, 30, 30]
```

Out[1]:
```
['rahul', 'vidhatri', 'sanika', 'vidhatri']
```

Out[1]:
```
[10, 20, 30, 30]
```

In [2]:
```
{'rahul', 'vidhatri', 'sanika', 'vidhatri'}
{10, 20, 30, 30, 30, 30}
```

Out[2]:
```
{'rahul', 'sanika', 'vidhatri'}
```

Out[2]:
```
{10, 20, 30}
```

In [4]:
```
names = ['rahul', 'vidhatri', 'sanika', 'vidhatri']
names[0]
names[1]
names[2]
names[3]
```

Out[4]:
```
'rahul'
```

Out[4]:
```
'vidhatri'
```

Out[4]:
```
'sanika'
```

Out[4]:
```
'vidhatri'
```

In [6]:
```
names = {'rahul', 'vidhatri', 'sanika', 'vidhatri'}
# names[0] not possible
names
```

Out[6]:
```
{'rahul', 'sanika', 'vidhatri'}
```

In [7]:
```
names = ['rahul', 'vidhatri', 'sanika', 'vidhatri']
names.append('cmd')

names
```

Out[7]:
```
['rahul', 'vidhatri', 'sanika', 'vidhatri', 'cmd']
```

In [9]:
```
names = ('rahul', 'vidhatri', 'sanika', 'vidhatri')
# names.append('cmd')
names
```

Out[9]:
```
('rahul', 'vidhatri', 'sanika', 'vidhatri')
```

to make a collection of values, you've 3 options

1. make a list (default choice)
2. make a set (use this if you don't want duplicates in collection, caution: no order/index)
3. make a tuple (use this if you don't want to modify the collection)

In [23]:
```
# list methods

L = [5, 8, 1, 3, 10, 5]

print(f'{L.index(5)=}')
print(f'{L.count(5)=}')

L.append(99)
L.remove(5)

L.sort()
print(f'{L=}')

L.reverse()
print(f'{L=}')
```

```
L.index(5)=0
L.count(5)=2
L=[1, 3, 5, 8, 10, 99]
L=[99, 10, 8, 5, 3, 1]
```

In [37]:
```
my_set = {5, 8, 1, 3, 10, 5}

my_set.add(99)
my_set.remove(8)  # remove value, but error if value not present
my_set.discard(18)  # remove only if value if present, otherwise pass, no error

print(f'{my_set=}')
```

```
my_set={1, 3, 99, 5, 10}
```

In [44]:
```
set_1 = {1, 2, 3, 4, 5, 6}
set_2 = {5, 6, 10}

set_1 | set_2  # union (or) [pipe]
set_1 & set_2  # intersection (and) [ampersand]
set_1 - set_2  # difference [hyphen]
set_1 ^ set_2  # symmetric difference (union - intersection) [caret]
```

Out[44]:
```
{1, 2, 3, 4, 5, 6, 10}
```

Out[44]:
```
{5, 6}
```

Out[44]:
```
{1, 2, 3, 4}
```

Out[44]:
```
{1, 2, 3, 4, 10}
```

In [45]:
```
set_1 = {1, 2, 3, 4, 5, 6}
set_2 = {5, 6, 10}

set_1.union(set_2)  # union (or) [pipe]
set_1.intersection(set_2)  # intersection (and) [ampersand]
set_1.difference(set_2)  # difference [hyphen]

# symmetric difference (union - intersection) [caret]
set_1.symmetric_difference(set_2)
```

Out[45]: {1, 2, 3, 4, 5, 6, 10}

Out[45]: {5, 6}

Out[45]: {1, 2, 3, 4}

Out[45]: {1, 2, 3, 4, 10}

In [47]:
```
tuple_1 = (10, 20, 30, 30, 30, 30)

tuple_1.index(10)
tuple_1.count(30)
```

Out[47]: 0

Out[47]: 4

In [53]:
```
L = [10, 20, 30, 30, 50, 10, 20, 40]

# duplicates must go away
# collection should not be modified later (immutability: the state of not changing)

tuple(set(L))
```

Out[53]: (40, 10, 50, 20, 30)

In [59]:
```
data = ('Vidhatri', 12, 50)

name, grade, weight = data # tuple unpacking (no. of vars on LHS must equal len of

print(name, grade, weight)
```

```
Vidhatri 12 50
```

In [87]:
```
data = {
    'name': 'vidhatri',  # key-value pair
    'age': 20,
    'best_friend': {
        'name': 'someone',
        'age': 19,
        'father': {
            'job': 'businessman',
            'salary': '10lacs'
        }
    },
    'fav_movies': ['abc', 'xyz', 'qwe', 'qwe', 'qwe', 'qwe'],
}

# data['best_friend']['father']['salary']
data['fav_movies']
```

Out[87]: ['abc', 'xyz', 'qwe', 'qwe', 'qwe', 'qwe']

In [112…
```
employee = {
    'name': 'john',
    'job': 'software developer',
```

```
    'department': 'backend',
    'age': 25,
    'salary': '15 lacs'
}

# employee.keys()  # keys of dict
# employee.values()  # all the values in dict
# employee.items()  # all key-value pairs/tuple

# for key in employee.keys():
#   print(f'key is {key}')

# for value in employee.values():
#   print(f'value is {value}')

# for key, value in employee.items():
#   print(f'key is {key}, value is {value}')

# dict_items [but we were hoping it to be of type `list`, WTH!!]
# dict_items: can't enjoy indexing or any list features 😨
# dict_items: but you still can loop over it [for key, value in dict.items()] 😃
# type(employee.items())

# converted to a list to enjoy indexing and other list benefits
# key, value = list(employee.items())[0]
# print(key, value)

# for key, value in employee.items():
#   print(key, value)

# print(type(employee.keys()))  # 🚨 not a `list`
# print(type(employee.values()))  # 🚨 not a `list`
# print(type(employee.items()))  # 🚨 not a `list`
```

In [126…

```
employee = {
    'name': 'john',
    'job': 'software developer',
    'department': 'backend',
    'age': 25,
    'salary': '15 lacs'
}

# key in employee <- this checks if key present in dict (T/F)
```

Accept a sequence of words as input. Create a dictionary named `freq` whose keys are the distinct words in the sequence. The value corresponding to a key (word) should be the frequency of occurrence of the key (word) in the sequence.

---

(1) You can assume that all words will be in lower case.

(2) You do not have to print the output to the console. This will be the responsibility of the autograder.

In [133…

```
words = ['a', 'b', 'c', 'a', 'd', 'a', 'b', 'c', 'e',
         'd', 'a', 'q', 't', 'v', 'z', 'x', 's', 'a', 'b', 'd', 's', 'e',
         'f', 'g', 'a', 'b', 'c', 'e', 't', 'q']
```

```
words = ['public', 'uber', 'private', 'uber',
         'public', 'uber', 'private', 'uber']

freq = {}

for word in words:
  if word not in freq:
    freq[word] = 0

  freq[word] += 1

freq
```

Out[133… {'public': 2, 'uber': 4, 'private': 2}