CLASS 6

LIST COMPREHENSIONS

List Comprehensions

- Essentially, it is Python's way of **implementing a well-known notation for sets as used by mathematicians**.

- In mathematics the square numbers of the natural numbers are, for example, created by $\{ x2 \mid x \in \mathbb{N} \}$ or the set of complex integers $\{ (x,y) \mid x \in \mathbb{Z} \wedge y \in \mathbb{Z} \}$.

- **List comprehension** is an **elegant way to define and create lists in Python**.

List Comprehensions

- Let us learn by example
- Create a list containing squares of natural numbers from 1 to 9 using list comprehension:

```
[x**2 for x in range(1,10)]
```

List Comprehension

```
x = []
for i  in range(1,10):
   x.append(i**2)
print(x)
```

Regular for loop

List Comprehensions

- Create a list containing squares of even natural numbers from 1 to 9 using list comprehension

```
# General way
x = []
for i  in range(1,10):
    if i%2 == 0:
        x.append(i**2)
print(x)
```

```
[4, 16, 36, 64]
```

Regular for loop and if loop

```
# Comprehension way  (Pythonic way)
[x**2 for x in range(1,10) if x%2 == 0]
```

```
[4, 16, 36, 64]
```

List Comprehension

List Comprehensions: Examples

- Create the Pythagorean triples in the range 1 to 30.

  Expected Output: [(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12, 15), (10, 24, 26), (12, 16, 20), (15, 20, 25), (20, 21, 29)]

```
[(x,y,z) for x in range(1,30)
         for y in range(x,30)
         for z in range(y,30)
         if x**2 + y**2 == z**2]
```

```
[(3, 4, 5),
 (5, 12, 13),
 (6, 8, 10),
 (7, 24, 25),
 (8, 15, 17),
 (9, 12, 15),
 (10, 24, 26),
 (12, 16, 20),
 (15, 20, 25),
 (20, 21, 29)]
```

List Comprehensions: Examples

- Create Cross product of two sets called coloured_things (as specified below):

  colours = ["red", "green", "yellow", "blue"]

  things = ["house", "car", "tree"]

  coloured_things (Combine the 2 lists)

```
colours = [ "red", "green", "yellow", "blue" ]
things = [ "house", "car", "tree" ]

[ (x,y) for x in colours for y in things ]
```

```
[('red', 'house'),
 ('red', 'car'),
 ('red', 'tree'),
 ('green', 'house'),
 ('green', 'car'),
 ('green', 'tree'),
 ('yellow', 'house'),
 ('yellow', 'car'),
 ('yellow', 'tree'),
 ('blue', 'house'),
 ('blue', 'car'),
 ('blue', 'tree')]
```

List Comprehensions: Examples

- Take the following paragraph

  paragraph = ["There was a fox." , 'It was brown in color.']

  Make a list of the words in the above paragraph using list comprehensions.

  Hint: Use split()

```
paragraph = ["There was a fox." , 'It was brown in color.']

single_word_list  = [word for sentence in paragraph for word in sentence.split()]
print(single_word_list)
```

```
['There', 'was', 'a', 'fox.', 'It', 'was', 'brown', 'in', 'color.']
```

List Comprehensions HW

1. In a school, there are total 20 students numbered from 1 to 20. You're given three lists named 'C', 'F', and 'H', representing students who play cricket, football, and hockey, respectively. Based on this information, find out and print the following:

Students who play all the three sports

Students who play both cricket and football but don't play hockey

Students who play exactly two of the sports

Students who don't play any of the three sports

Format:

Input:

3 lists containing numbers (ranging from 1 to 20) representing students who play cricket, football and hockey respectively.

Output:

4 different lists containing the students according to the constraints provided in the questions.

Note: Make sure you sort the final lists (in an ascending order) that you get before printing them; otherwise your answer might not match the test-cases.

Previous problem continued

- **Examples:**
  Input 1:
  [2, 5, 9, 12, 13, 15, 16, 17, 18, 19]
  [2, 4, 5, 6, 7, 9, 13, 16, 20]
  [1, 2, 5, 9, 10, 11, 12, 13, 15, 20]
  Output 1:
  [2, 5, 9, 13]
  [16]
  [12, 15, 16, 20]
  [3, 8, 14]

  Explanation:
  1.Given the three sets, you can see that the students numbered '2', '5', '9', and '13' play all the three sports.
  2. The student numbered '16' plays cricket and football but doesn't play hockey.
  3. The students numbered '12' and '15' play cricket and hockey, the student numbered '16' plays cricket and football and the student numbered '20' plays football and hockey. Hence, the students who play exactly two sports are 12, 15, 16, and 20.
  4. As you can see, the students who play none of the sports are 3, 8, and 14.

- Do all the problems in conditionals using list comprehensions.

DICTIONARIES

Dictionaries

Until now, We are storing using separate lists and tuples for every information

Example:

List of names

names = ['Ana','John','Denise','Katie']

Grades of the above mentioned names

grades = ['B','A+','A','A']

- A **separate list** for each item.
- Each list must have the **same length**
- Info stored across lists at same index. Each index refers to one distinct person

Dictionaries

- There is a way to store all the information related to one person in the above example at one place using **"Dictionaries".**
- **Dictionaries** store in **pairs of data**.
  - **Key** (example: Name of a person)
  - **Value** (example: Grades obtained)

'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'katy'	'A'
Keys: These are custom index by label	**values:** Elements of that particularlabel

Representation of Dictionary: **{}**

Empty_dict = {}

Dictionaries: Example

```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
```

Key 1 value 1 Key 2 value 2 Key 3 value 3 Key 4 value 4

Dictionaries: Lookup

- similar to indexing into a list
- **looks up the key**
- **returns** the **value** associated with the key
- if key isn't found, get an error

'Ana'	'B'
'Denise'	'A'
'John'	'A+'
'katy'	'A'

```
grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}
print(grades['John'])
```

```
A+
```

```
grades['Sylvan']
```

Try this?

Dictionaries Operations

Example: grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}

- **Add** an entry
  grades[‘Sylvan’] = ‘A’ → **Output:**

‘Ana’	‘B’
‘Denise’	‘A’
‘John’	‘A+’
‘katy’	‘A’
‘Sylvan’	‘A’

- **Test if key** in dictionary
  ‘John’ in grades # **Output:** True
  ‘Daniel’ in grades # **Output:** False

- **Delete** entry
  Del(grades[‘Ana’]) → **Output:**

‘Denise’	‘A’
‘John’	‘A+’
‘katy’	‘A’
‘Sylvan’	‘A’

Dictionaries Operations

Example: grades = {'Ana':'B', 'John':'A+', 'Denise':'A', 'Katy':'A'}

‘Ana’	‘B’
‘Denise’	‘A’
‘John’	‘A+’
‘katy’	‘A’

No guaranteed order

- Tuple of all keys

  grades.keys()

  Output: dict_keys(['Denise', 'Katy‘, ’John’ , ’Ana’])

- Tuple of all values

  grades.values()

  Output: dict_values([A', ‘A', ‘A+‘, ’B’])

Dictionaries: Keys and Values

- **values**
  - any type (immutable and mutable)
  - can be duplicates
  - dictionary values can be lists, even other dictionaries!
- **keys**
  - must be unique
  - immutable type (int, float, string, tuple,bool)
  - actually need an object that is hashable, but think of as immutable as all immutable types are hashable
  - careful with float type as a key

List Vs Dictionary

LIST	DICT
• **ordered** sequence of elements	• **Matches** keys to values
• look up elements by integer index	• Looks up one item by another item
• Indices have an **order**	• **No order guaranteed**
• **Index is an integer**	• **Index here is called key** and it can be any **immutable type**

Dictionaries: Examples

- Create and print a dictionary:

```
thisdict = {
            "brand": "Ford",
            "model": "Mustang",
            "year": 1964
                }
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Both syntax work the same way

Note: *The keywords are not string literals*

The use of equals rather than colon for the assignment

Dictionaries: Examples

- You can access the items of a dictionary by referring to its key name, inside square brackets:

```
x = thisdict["model"]
print(x)
```

```
Mustang
```

- There is also a method called **get()** that will give you the same result:

```
x = thisdict.get("model")
print(x)
```

```
Mustang
```

Both syntax work the same way

Dictionaries: Examples

- You can change the value of a specific item by referring to its key name:

```
thisdict = { "brand": "Ford", "model": "Mustang", "year": 1964 }
```

```
thisdict["year"] = 2018
```

- Try and print **thisdict**

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': **2018**} ← value in **year changed**

Dictionaries: Examples

- "for" loop on dictionaries
  - You can loop through a dictionary by using a for loop.

```
for x in thisdict:
  print(x)
```

What will be the output?

Output:
brand
model
year

Prints all the keys in the dictionary

Dictionaries: Examples

- Print all values in the dictionary, one by one:

```
for x in thisdict:
  print(thisdict[x])
```

Or

- Use the values() function to return values of a dictionary:

```
for x in thisdict.values():
  print(x)
```

What will be the output?

Output:
Ford
Mustang
2018

Prints all the values in the dictionary

Dictionaries: Examples

- Loop through both keys and values, by using the items() function:

```
for x, y in thisdict.items():
    print(x, y)
```

Output:
brand Ford
model Mustang
year 2018

Item in dictionary refers to set of **(key, value)**

Dictionaries: Examples

- To determine how many items (key-value pairs) a dictionary have, use the len() method.

```
print(len(thisdict))
```

- Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict = { "brand": "Ford", "model": "Mustang",
"year": 1964 }
thisdict["color"] = "red"
print(thisdict)
```

Dictionaries: Examples

The clear() keyword empties the dictionary:

```
thisdict = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
thisdict.clear()
print(thisdict)
```

- **It empties the dictionary. Dictionary will not have any items.**

Dictionaries: Examples

- The del keyword can also delete the dictionary completely:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
del (thisdict)
print(thisdict)
```

- **This will cause an error because "thisdict" no longer exists.**

Dictionary Exercise:

1. Write a Python script to check if a given key already exists in a dictionary
2. Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).
3. Write a Python e program to sum all the items in a dictionary.
4. Write a Python program to map two lists into a dictionary.
5. Write a Python program to get the maximum and minimum value in a dictionary.
6. Write a Python program to combine two dictionary adding values for common keys.
7. Write a Python program to print all unique values in a dictionary.
   Sample Data : [{"V":"S001"}, {"V": "S002"}, {"VI": "S001"}, {"VI": "S005"}, {"VII":"S005"}, {"V":"S009"},{"VIII":"S007"}]
   Expected Output : Unique Values: {'S005', 'S002', 'S007', 'S001', 'S009'}
8. Write a Python program to combine values in python list of dictionaries.