

# Intro to Coding

Class 5

# Lists

- Used to store multiple values in a single variable
- These values can be different data types. However, it is good practice to keep all values in the list as the same data type
- Indices in lists are similar to the ones used in strings: they are zero indexed (first index starts at **0**)
- An entire list can be printed using `print(list_name)`

```
integers = [1, 2, 3, 4, 5, 6, 7]
```

```
print(integers[1:4])
```

What do you think will be printed?

# List Indexing

- Once again, similar to lists, ranges can be formed using the **[start:end:step]** syntax
  - Start: the starting index
  - End: the end index (be careful: end is non inclusive, ie. it will not include the end in the range!)
  - Step: the increments in-between the ranges. For example: **2** will result in every other value being included

```
ints = [1, 2, 3, 4, 5, 6, 7]

print(ints[3:])
print(ints[-1])
print(ints[2])
print(ints[0])
print(ints[0:7:2])
```

# List Functions

- `len(list_name)` - returns the length of the list
  - In other words, the number of elements in the list
- `list()` - creates an empty list
  - Alternatively, you can write `list_name = []` to create an empty list
- `list_name.append(elem)` - adds an element to **the end** of the list
- `list_name.pop(idx)` - removes the element at the **specified index**
  - If no index is specified, it will remove the **last element** instead
- `list_name.insert(idx, elem)` - inserts an element at the **specified index**
- `list_name.remove(elem)` - removes the **first instance** of the specified element
- `if elem in list_name:` - if statement to check if an element is in the list

## List Activity (Part 1)

Write a program that will continuously take in user input, and appending it to a list. The program will stop taking user input once the user inputs "QUIT". Then, print the list.

```
hello  
codeup  
quit  
quit  
QUIT  
['hello', 'codeup', 'quit', 'quit']
```

QUIT

[]

## Part 1 Solution

```
word = input()
word_list = []
while word != "QUIT":
    word_list.append(word)
    word = input()

print(word_list)
```

## List Activity (Part 2)

Modify your program from part 1 so that after the **first** instance of QUIT, the program will continuously take in user input, removing the **first instance** of it from the list. Then, print the list after the removals.

Note: make sure to remove the print statement from part 1

```
a
b
c
QUIT
d
QUIT
['a', 'b', 'c']
```

```
a
b
a
a
QUIT
a
a
QUIT
['b', 'a']
```

## Part 2 Solution

```
word = input()
word_list = []
while word != "QUIT":
    word_list.append(word)
    word = input()

word = input()
while word != "QUIT":
    if word in word_list:
        word_list.remove(word)
    word = input()

print(word_list)
```



## Additional List Functions

- `list_name.sort()` - sorts the elements in the list
  - If it is an **integer** list, it sorts elements in **ascending** order (eg. `[1, 3, 5, 6, 7, 7]` )
  - If it is a **string** list, it sorts elements in **alphabetical** order (eg. `["apple", "ban", "banana", "ora", "oran", "orange", "orango"]` )
- `list_name.reverse()` - reverses the elements in the list
  - eg. `[1, 2, 3, 4, 5] → [5, 4, 3, 2, 1]`

# Iterating Through a List

- There are two ways to iterate through a list
- This method involves iterating in a range and getting an element at the index

```
num_list = [1, 2, 3, 4, 5]
for i in range(len(num_list)):
    print(num_list[i])
```

- This method involves directly looping the elements of this list, does not require getting the value at the index

```
num_list = [1, 2, 3, 4, 5]
for i in num_list:
    print(i)
```

## Concatenating Lists

- Lists can be concatenated in a similar manner to string concatenation

```
a = [1, 2, 3]
b = ["a", "b", "c"]
print(a + b)
```

```
[1, 2, 3, 'a', 'b', 'c']
```

# List Activity

Write a program that will continuously take in **integer** user inputs, appending it to a list. The program will stop taking in user input once the user enters “-1”. Then, sort the list in **descending order**. Starting from the first element, add 10 to every **third element** after it (ie. 1st, 4th, 7th, ...)

```
1
3
5
3
9
-3
-7
-1
Sorted list: [9, 5, 3, 3, 1, -3, -7]
Final list: [19, 5, 3, 13, 1, -3, 3]
```

# Solution

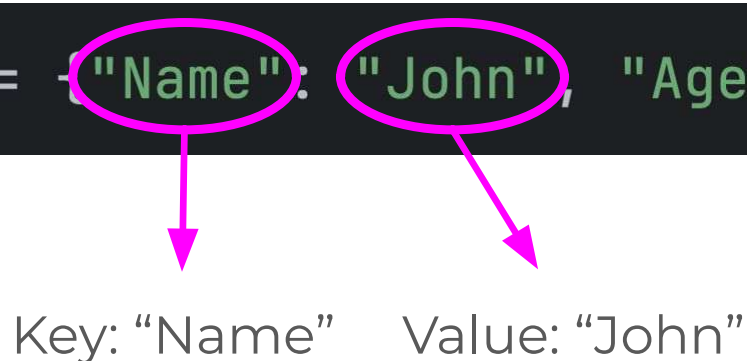
```
num = int(input())
num_list = []
while num != -1:
    num_list.append(num)
    num = int(input())

num_list.sort()
num_list.reverse()
print("Sorted list: " + str(num_list))
for i in range(0, len(num_list), 3):
    num_list[i] += 10
print("Final list: " + str(num_list))
```

# Dictionaries

- A data structure that stores multiple values, similar to a list
- Unlike lists, dictionaries follow a (key, value) pair structure
  - Similar to how real dictionaries follow a (word, definition) structure
- Dictionaries are defined with curly brackets {}, and the pairs are defined in the format `key: value`, separated by commas

```
d = {"Name": "John", "Age": "40", "Job": "Doctor"}
```



Key: "Name"    Value: "John"

## Accessing Values

- Values can be accessed similar to lists using square brackets [], however you put the key in the square brackets rather than the index. **Indices do not exist in dictionaries!**

```
d = {"Name": "John", "Age": "40", "Job": "Doctor"}  
print("Hello, " + d["Name"])
```

- Alternatively, you can use the .get() function

```
d = {"Name": "John", "Age": "40", "Job": "Doctor"}  
print("Hello, " + d.get("Name"))
```

## Printing Dictionaries

- Dictionaries can be printed by doing `print(dict_name)`
- This prints it in a similar manner to lists

```
d = {"Name": "John", "Age": "40", "Job": "Doctor"}  
print(d)
```

```
{'Name': 'John', 'Age': '40', 'Job': 'Doctor'}
```



# Lists in Dictionaries

- Lists are valid objects that can be put in dictionaries

```
d = {"Numbers": [1, 12, -3, 0], "Letters": ["a", "b", "c", "d"]}
```

- In the following example, the **key** Numbers is mapped to the corresponding **value** [1, 12, -3, 0]
- You can loop through a list in a dictionary, shown in the code below

```
d = {"Numbers": [1, 12, -3, 0], "Letters": ["a", "b", "c", "d"]}
for i in range(len(d["Numbers"])):
    print(d["Numbers"][i])
```

Looping using an index

Directly looping through  
the objects

```
d = {"Numbers": [1, 12, -3, 0], "Letters": ["a", "b", "c", "d"]}
for i in d["Numbers"]:
    print(i)
```

# Dictionary Activity

- The dictionary contains two key-pair values
  - Students - a list containing the students' names
  - Marks - a list containing the students' marks
- The student at index *i* has a corresponding mark at index *i* (ie. Adam has a 90, Bob has a 96, etc.)

```
d = {"Students": ["Adam", "Bob", "Cam", "Dan"], "Marks": [90, 96, 86, 100]}
```

Your task: add to the program to replicate the output below

```
The highest scoring student is Dan with a mark of 100  
The class average is 93.0
```

# Solution

We need a variable `high_idx` to track the index of the high score. This index is needed to find the highest scoring student's name. We update anything related to the highest mark whenever the new element is greater than the highest mark so far.

```
d = {"Students": ["Adam", "Bob", "Cam", "Dan"], "Marks": [90, 96, 86, 100]}
high = 0
student = ""
total = 0
for i in range(len(d["Marks"])):
    if d["Marks"][i] > high:
        high = d["Marks"][i]
        student = d["Students"][i]
    total += d["Marks"][i]

print("The highest scoring student is " + student + " with a mark of " + str(high))
print("The class average is " + str(total / len(d["Marks"])))
```

# Dictionary Functions

`dict_name.update(key: value)` - adds a new key-value pair if the key doesn't exist already. If the key exists in the dictionary, it will update the corresponding value in the key

`dict_name.pop(key)` - removes key, and its corresponding value from the dictionary

```
d = {}  
d.update({"Name": "John"})  
d.update({"Age": 30})  
d.pop("Name")
```

Adds the key-value pair "Name": "John"

Removes the key-value pair "Name": "John"

## Dictionary Properties

Alternatively, we can achieve the same effect with the following properties of dictionaries. We can modify the key directly by setting its key, and delete it through the `del` keyword

```
d = {}  
d["Name"] = "John"  
d["Age"] = 30  
del d["Name"]
```

## A List of Dictionaries

You can have a list that stores dictionaries.

```
dict_list = [{"Name": "John", "Age": 30}, {"Name": "Dave",  
"Age": 45}]
```

```
dict_list = [{"Name": "John", "Age": 30}, {"Name": "Dave", "Age": 45}]
```

## Activity

Using your previous code, modify it so that it contains a list of dictionaries instead. This means that each index  $i$  of the list will contain a dictionary that stores the student's name and mark.

# Solution

Each element (ie. dictionary) contains all of student *i*'s information

```
dict_list = [{"Name": "Adam", "Mark": 90}, {"Name": "Bob", "Mark": 96}, {"Name": "Cam", "Mark": 86}, {"Name": "Dan", "Mark": 100}]

high = 0
student = ""
total = 0
for i in dict_list:
    if i["Mark"] > high:
        high = i["Mark"]
        student = i["Name"]
    total += i["Mark"]

print("The highest scoring student is " + student + " with a mark of " + str(high))
print("The class average is " + str(total / len(dict_list)))
```



## Activity 2

Write a program that will continuously prompt the user to enter the following information: name, age, profession

The program should stop taking in input after the user inputs all of Jack's information, and then print everyone's information.

```
Enter the person's name: John  
Enter the person's age: 40  
Enter the person's profession: Doctor  
Enter the person's name: Bob  
Enter the person's age: 30  
Enter the person's profession: Teacher  
Enter the person's name: Jack  
Enter the person's age: 10  
Enter the person's profession: Unemployed
```

```
Information for person 1:  
Name: John  
Age: 40  
Profession: Doctor  
Information for person 2:  
Name: Bob  
Age: 30  
Profession: Teacher  
Information for person 3:  
Name: Jack  
Age: 10  
Profession: Unemployed
```

# Solution

We can use a while loop to continuously take in input. Then, we use an indexed for loop to output everyone's information. We add **1** to the index when printing the person's number because lists are 0-indexed.

```
name = ""
age = 0
prof = ""
dict_list = []
while name != "Jack":
    name = input("Enter the person's name: ")
    age = int(input("Enter the person's age: "))
    prof = input("Enter the person's profession: ")
    dict_list.append({"Name": name, "Age": age, "Profession": prof})

for i in range(len(dict_list)):
    print("Information for person " + str(i + 1) + ":")
    print("Name: " + dict_list[i]["Name"])
    print("Age: " + str(dict_list[i]["Age"]))
    print("Profession: " + dict_list[i]["Profession"])
```

# Homework

You have been tasked to create a program that handles account sign-ups for a company. Each account should have the name of the user, their birth-year, their email, their username, and their password.

However, your program must also be able to handle more than 1 sign-up, thus you must store all these users somehow.

If the user inputs their birth-year as  $< 1930$ , end the program and print out all the currently signed-up users. The user with the birth-year  $< 1930$  should **not** be saved.

# Homework

## Sample test cases

```
What is your name? anderson lai
What is your birth year? 2008
What is your email? anderson.lai@gmail.com
What is your username? anderson_lai
What is your password? hello!
```

```
What is your name? jeffrey smith
What is your birth year? 1999
What is your email? jeffrey@hotmail.com
What is your username? jeffrey1999
What is your password? apple_pie
```

```
What is your name? john white
What is your birth year? 1800
```

```
Welcome anderson lai. You were born in 2008, and your email is anderson.lai@gmail.com. Your username is anderson_lai and your password is hello!
Welcome jeffrey smith. You were born in 1999, and your email is jeffrey@hotmail.com. Your username is jeffrey1999 and your password is apple_pie
```