CS65: Introduction to Computer Science

List of List
Assignment 3
Dictionary

Md Alimoor Reza
Assistant Professor of Computer Science

Review: List

- List manipulation
  - Slicing
  - Appending
  - Updating
  - Removing
  - Other methods/operations
- List of lists
  - Understanding the dimensions
  - Accessing elements
  - Accessing with nested for loops

Review: List Slicing

- Format: *list_variable_name*[**start : end**]
  - **start :** starting index — if not specified 0 is used
  - **end :** end index — if not specified ***len****(list)* is used

```
num_list = [10, 11, 12, 13, 14, 15]

print("num_list ", num_list)
print("num_list[0] ", num_list[0])
print("num_list[:1] ", num_list[:1])
print("num_list[1:4] ", num_list[1:4])
print("num_list[1:] ", num_list[1:])
```

```
>>> %Run lec12.py

  num_list  [10, 11, 12, 13, 14, 15]
  num_list[0]  10
  num_list[:1]  [10]
  num_list[1:4]  [11, 12, 13]
  num_list[1:]  [11, 12, 13, 14, 15]
```

Review: Inserting Items in a List

- Appending elements in an empty list with **. append()** method
  - It is executed by a "**.**" (dot) symbol followed the method name

Sequentially appending items

```
# building list with append() function
num_list = []

num_list.append(2)
print("num_list: ", num_list)

num_list.append(4)
print("num_list: ", num_list)

num_list.append(6)
print("num_list: ", num_list)
```

```
>>> %Run lec11.py

  num_list:  [2]
  num_list:  [2, 4]
  num_list:  [2, 4, 6]
```

Appending items with for loop

```
    # building list with append function using loop
    # insert only multiples of 5
    lower_limit = int(input("enter the lower limit: "))
    upper_limit = int(input("enter the upper limit: "))

    num_list = []


    for num in range(lower_limit, upper_limit):

        if (num % 5 == 0):

            num_list.append(num)

    print("num_list: ", num_list)
```

```
>>> %Run lec11.py

  enter the lower limit: 3
  enter the upper limit: 22
  num_list:  [5, 10, 15, 20]

>>> %Run lec11.py

  enter the lower limit: 5
  enter the upper limit: 49
  num_list:  [5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Review: Changing Items in a List

- Changing specific items in a list

Sequentially updating items

```
num_list = [10, -1, -2, -3, 20, -4, 30]


print("Before modification num_list is ", num_list)


num_list[1] = 0
num_list[2] = 0
num_list[3] = 0
num_list[4] = 40
```

```
>>> %Run lec12.py

  Before modification num_list is  [10, -1, -2, -3, 20, -4, 30]
  After modification num_list is  [10, 0, 0, 0, 40, -4, 30]
```

Updating items with for loop

```
# modifying list based on a criteria

num_list = [10, -1, -2, -3, 20, -4, 30]

list_size = len(num_list)

print("Before modification num_list is ", num_list)
for i in range(list_size):

    if (num_list[i] < 0):

        #print("Found negative num at index: ", i)

        num_list[i] = 0

print("After modification num_list is ", num_list)
```

```
>>> %Run lec11.py

  Before modification num_list is  [10, -1, -2, -3, 20, -4, 30]
  After modification num_list is  [10, 0, 0, 0, 20, 0, 30]
```

Review: Removing Items from a List

Version 1: .remove() method

```
num_list = [10, -2, -2, -2, 20, -4, 30]

num_list.remove(-2)
num_list.remove(-2)
num_list.remove(-2)
print("After removing all -2s num_list is ", num_list)
```

```
>>> %Run lec12.py

  After removing all -2s num_list is  [10, 20, -4, 30]
```

Version 2: del keyword

```
# deleting an item from the list

num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

del num_list[0]

print("After removing item at index 0 num_list is ", num_list)

del num_list[1]

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Version 3: .pop() method

```
num_list = [10, -1, -2, -3, 20, -4, 30]

print("Initial num_list is ", num_list)

num_list.pop(0)

print("After removing item at index 0 num_list is ", num_list)

num_list.pop(1)

print("After removing item at index 1 num_list is ", num_list)
```

```
>>> %Run lec12.py

  Initial num_list is  [10, -1, -2, -3, 20, -4, 30]
  After removing item at index 0 num_list is  [-1, -2, -3, 20, -4, 30]
  After removing item at index 1 num_list is  [-1, -3, 20, -4, 30]
```

Other Useful List Operations

operation	meaning	result type
x in s	checks if an item in s equals x	bool
x not in s	checks if no items in s equal x	bool
s + t	concatenation (two sequences)	same seq. type
s*n (or: n*s)	n shallow copies of s, concatenated	same seq. type
s.count(x)	find # items in s equal to x	int (#matches)
s.index(x)	find index of first x in s (if not found, crashes)	int

Other Useful List Operations

- **list1 + list2** produces new list by concatenating list2 to end of list1
- **min(list)** finds the elements in the list with the smallest value
- **max(list)** finds the elements in the list with the largest value

operation	meaning	returned value
s.append(x)	add x as a single value at end of s.	None value
s.extend(t)	individually append each item of sequence t to the end of s.	None value
s.insert(i,x)	make space (push other spots to the right), put x value at location i.	None value
s.pop(i)	remove value at index i from sequence; return the value that was there	item that was at index i
s.remove(x)	find first occurrence of x, remove it.	None
s.reverse()	reverse the ordering of items.	None

Poll

- Follow the link below and submit your answer

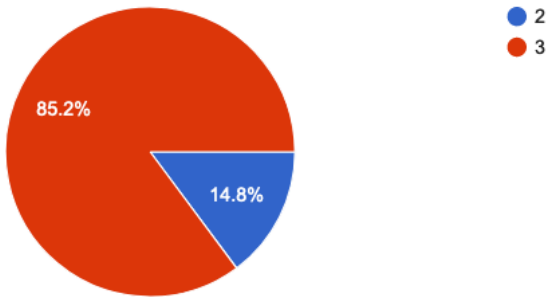
https://tinyurl.com/mt63pm96

Review: Poll

What is the length of the list [100, 200, 400]?

Copy

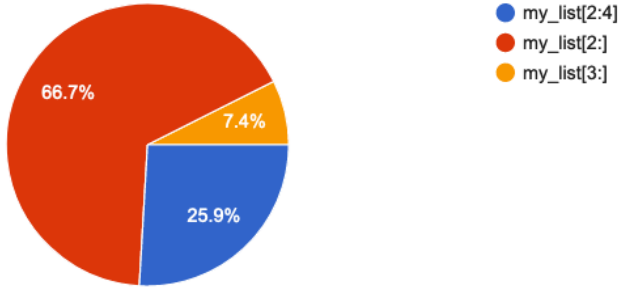
27 responses



Slice is a span of items that are taken from a List or any sequence. Let's denote my_list = [1, -4, 3, 5, 6]. Use slicing operation to get 3,5,6 from my_list.

Copy

27 responses

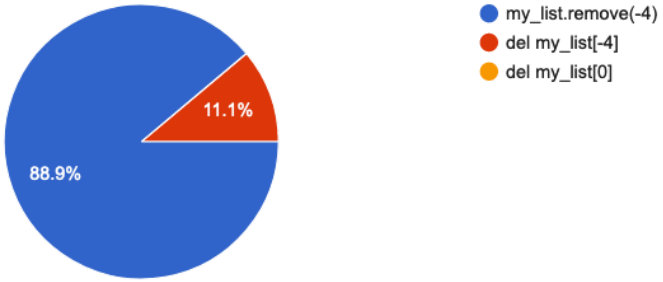


Review: Poll

Let's denote my_list = [1, -4, 3, 5, 6]. How can you remove item -4 from my_list?

Copy

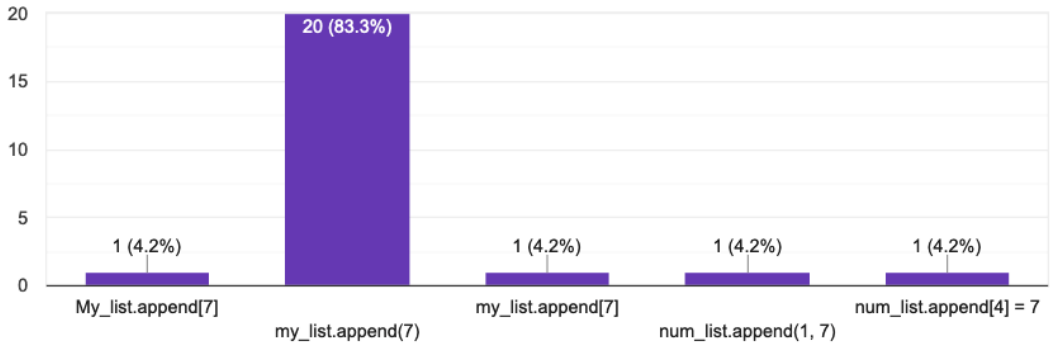
27 responses



Add number 7 at the end of my_list = [1, -4, 3, 5, 6]? Hint: You can use .append() method.

Copy

24 responses

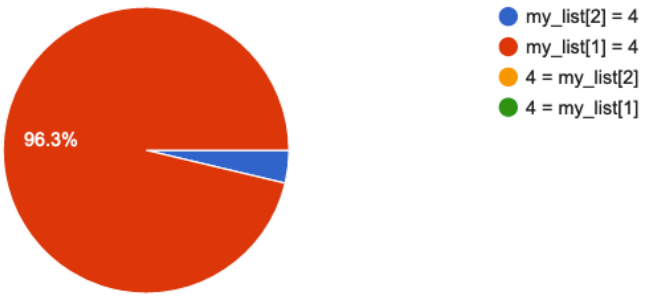


Review: Poll

my_list = [1, -4, 3, 5, 6]. What would be operation to replace -4 with 4 in this list?

Copy

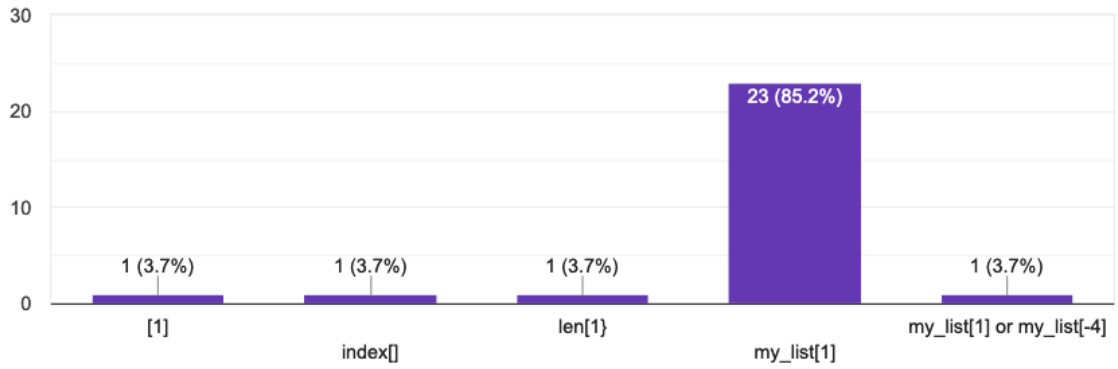
27 responses



Let's denote a variable my_list = [1, -4, 3, 5, 6]. How can you access the -4? Hint: use []

Copy

27 responses



Understanding the 'List of Lists'

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]
```

- **Step 1:** Understand each dimension

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]

list_size_outer  = len(num_list)
print("Size of the outer list ", list_size_outer)

list_size_inner0 = len(num_list[0])
print("Size of the first inner-list ", list_size_inner0)

list_size_inner1 = len(num_list[1])
print("Size of the second inner-list ", list_size_inner1)
```

```
>>> %Run lec12.py

  Size of the outer list  2
  Size of the first inner-list  3
  Size of the second inner-list  3
```

Understanding the 'List of Lists'

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]
```

- **Step 2:** Access element in each dimension
  - outer dimension

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]

print("1st element in outer-list ", num_list[0])
print("2nd element in outer-list ", num_list[1])
```

```
>>> %Run lec12.py

  1st element in outer-list  [1, 2, 3]
  2nd element in outer-list  [10, 20, 30]
```

Understanding the 'List of Lists'

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]
```

- **Step 2:** Access element in each dimension
  - inner dimension

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]

print("1st  element in inner-list0 ", num_list[0][0])
print("2nd  element in inner-list0 ", num_list[0][1])
print("3rd  element in inner-list0 ", num_list[0][2])
```

```
>>> %Run lec12.py

  1st  element in inner-list0  1
  2nd  element in inner-list0  2
  3rd  element in inner-list0  3
```

Recap: Nested **for** loops

- Putting one loop inside another
  - The first loop is called the outer loop
  - The second loop is called the inner loop

- Here is simpler version:

```
for i in range(3):
    for j in range(3):
        print("i: ", i, "j: ", j)
```

Recap: Visualization of nested **for** loop

```
# nested for loop
for i in range(2):
    # first segment inside outer loop
    for j in range(3):
        # first segment inside inner loop
        print("i ->", i, " j ->", j)
        # next segment inside inner loop

        # ... segment insdie inner loop

    # next segment inside outer loop

    # ... segment inside outer loop
```

Notice the alignment (inner-loop)

Notice the alignment (outer-loop)

List of Lists and Nested Index Loops

```
# list of lists

num_list = [ [1, 2, 3], [10, 20, 30] ]
```

- **Step 1:** Create an index for each dimension
- **Step 2:** Nest loops
- **Step 3:** Access each element using indexing

List of Lists and Nested Index Loops

```
# list of lists
num_list = [ [1, 2, 3], [10, 20, 30] ]
list_size_i = len(num_list)
for i in range(list_size_i):
    inner_list = num_list[i]      # get one inner list
    list_size_j = len(inner_list) # find the size of the inner list
    for j in range(list_size_j):
        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py
  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

List of Lists and Nested Index Loops

- Code simplification
  - Inserting the *len()* inside the range function

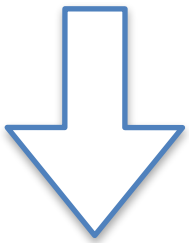
```
num_list = [ [1, 2, 3], [10, 20, 30] ]
for i in range(len(num_list)):
    for j in range(len(num_list[i])):
        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py
num_list[ 0 ][ 0 ] 1
num_list[ 0 ][ 1 ] 2
num_list[ 0 ][ 2 ] 3
num_list[ 1 ][ 0 ] 10
num_list[ 1 ][ 1 ] 20
num_list[ 1 ][ 2 ] 30
```

Exercise 3

- Find the summation of all the numbers in a list of lists

```
num_list = [ [1, 2, 3], [10, 20, 30] ]
```



```
>>> %Run lec12.py
  Total is  66
```

```
num_list = [ [1, 2, 3], [10, 20, 30] ]
for i in range(len(num_list)):
    for j in range(len(num_list[i])):
        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py
  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

List Slicing

- Slice is a span of items that are taken from a list (or any sequence)
  - Span is a list containing copies of elements from **start** up to, but not including, **end**
  - Format: *list_variable_name*[**start : end**]
    - **start :** starting index — if not specified 0 is used
    - **end :** end index — if not specified ***len****(list)* is used

List Slicing

- Format: *list_variable_name*[**start : end**]
  - **start :** starting index — if not specified 0 is used
  - **end :** end index — if not specified ***len****(list)* is used

```
num_list = [10, 11, 12, 13, 14, 15]

print("num_list ", num_list)
print("num_list[0] ", num_list[0])
print("num_list[:1] ", num_list[:1])
print("num_list[1:4] ", num_list[1:4])
print("num_list[1:] ", num_list[1:])
```

```
>>> %Run lec12.py

num_list  [10, 11, 12, 13, 14, 15]
num_list[0]  10
num_list[:1]  [10]
num_list[1:4]  [11, 12, 13]
num_list[1:]  [11, 12, 13, 14, 15]
```

Other Useful Slicing Operations

operation	meaning
s[i] = x	replace ith item of s with x
s[i:j] = t	replace slice i:j with sequence t. (lengths needn't match!)
s[i:j:k] = t	replace slice i:j:k with sequence t. (lengths *must* match!)
del s[i]	remove ith item from s.
del s[i:j]	remove slice i:j from s.
del s[i:j:k]	remove slice i:j:k from s.

Review: Accessing the 'List of Lists'

- Access element in each dimension
  - outer dimension
  - inner dimension

```
print("1st element in outer-list ", num_list[0])
print("2nd element in outer-list ", num_list[1])
```

```
>>> %Run lec12.py

  1st element in outer-list  [1, 2, 3]
  2nd element in outer-list  [10, 20, 30]
```

```
print("1st  element in inner-list0 ", num_list[0][0])
print("2nd  element in inner-list0 ", num_list[0][1])
print("3rd  element in inner-list0 ", num_list[0][2])
```

```
>>> %Run lec12.py

  1st  element in inner-list0  1
  2nd  element in inner-list0  2
  3rd  element in inner-list0  3
```

- **Step 1:** Create an index for each dimension
- **Step 2:** Nest loops
- **Step 3:** Access each element using indexing

Review: List of Lists and Nested Index Loops

Version 1 (more lines but easy to understand)

```
num_list = [ [1, 2, 3], [10, 20, 30] ]

list_size_i = len(num_list)

for i in range(list_size_i):

    inner_list = num_list[i]      # get one inner list

    list_size_j = len(inner_list) # find the size of the in

    for j in range(list_size_j):

        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py

  num_list[ 0 ][ 0 ] 1
  num_list[ 0 ][ 1 ] 2
  num_list[ 0 ][ 2 ] 3
  num_list[ 1 ][ 0 ] 10
  num_list[ 1 ][ 1 ] 20
  num_list[ 1 ][ 2 ] 30
```

Version 2 (less lines of code)

```
num_list = [ [1, 2, 3], [10, 20, 30] ]

for i in range(len(num_list)):

    for j in range(len(num_list[i])):

        print("num_list[", i, "][", j, "]", num_list[i][j])
```

```
>>> %Run lec12.py

 num_list[ 0 ][ 0 ] 1
 num_list[ 0 ][ 1 ] 2
 num_list[ 0 ][ 2 ] 3
 num_list[ 1 ][ 0 ] 10
 num_list[ 1 ][ 1 ] 20
 num_list[ 1 ][ 2 ] 30
```

Assignment 3

- Post-midterm — focus will be more on assignments + final project
  - Assignments 3, 4 and final project

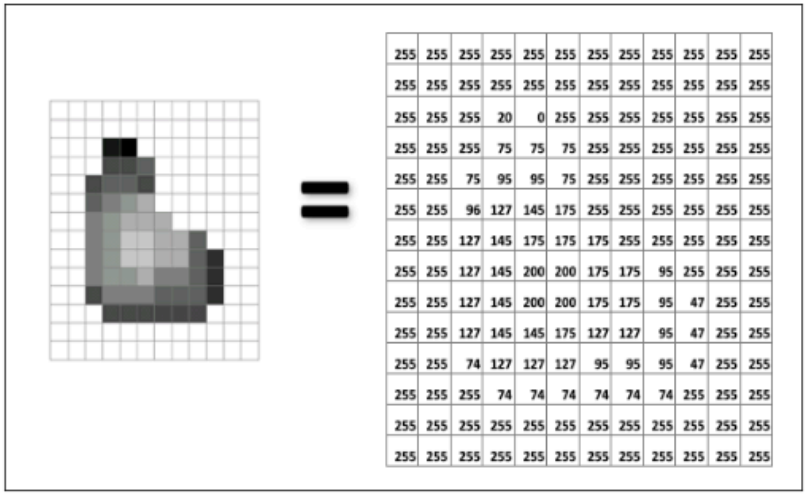
Assignment 3

- Task 1 (List manipulation)
  - Removing invalid items from the color-triplet
- Task 2 (Dictionary manipulation)
  - Finding the mapping

Assignment 3: Task 1 (List manipulation)

Subtask 3 (30 points): Color Correction

Computer vision researchers (including myself) analyze images to find meaningful content in them. An image is commonly represented by a three-channel grid matrix containing integer numbers. The figure below shows one such channel of an image. Each integer number represents an intensity value. These intensity values ranges between **0** and **255**. A large number signifies a brighter intensity, while a low number corresponds to a darker intensity.



A toy image and computer's internal representation as a grid of intensity values.

Let us assume you are given these intensity values as list of triplets (red, green, blue). Unfortunately, some of these triples got corrupted and values outside the valid range (between 0 and 255). Your program will receive a list of lists. Each inner list is a triplet representing three color values: one for red intensity, one for green intensity, and finally one for the blue intensity. For example, [[100, 0, 0], [40, -156, 0], [0, 156, 0], [40, 156, 500], [0, 0, 250]] represents a list of five color-triplets. **You should write three separate functions as instructed below.** Each function will perform a separate operation and accordingly will return a different list.

Assignment 3: Task 1 (List manipulation)

- *get_invalid_colors()*: If any intensity value inside a triplet has either a negative value (less than zero) or a positive value greater than 255, you should return the indices of those triplets inside the list. For example, given an input list of [[100, 0, 0], [40, -156, 0], [0, 156, 0], [40, 156, 500], [0, 0, 250]], your program should return list **[1, 3]** since indices at 1 and 3 contain two color-triplets with either a negative intensity or intensity value greater than 255.

- *correct_invalid_colors()*: If any intensity value inside a color-triplet has a negative value, you should replace that value with **0**. On the other hand, if any intensity value inside a color-triplet has a positive value greater than 255, you should replace that value with **255**. For example, given an input list of [[100, 0, 0], [40, -156, 0], [0, 156, 0], [40, 156, 500], [0, 0, 250]], your program should return [[100, 0, 0], [40, **0**, 0], [0, 156, 0], [40, 156, **255**], [0, 0, 250]].

- *discard_invalid_colors()*: If any intensity value inside a triplet has either a negative value (less than zero) or a positive value greater than 255, you should remove that color-triplet. For example, given an input list of [[100, 0, 0], [40, -156, 0], [0, 156, 0], [40, 156, 500], [0, 0, 250]], your program should return [[100, 0, 0], [0, 156, 0], [0, 0, 250]].

Assignment 3: Task 1 (List manipulation)

```
# this fucntion returns the list of indices with invalid color-triplets
def get_invalid_colors(my_list):


    return

# this fucntion returns the corrected list of color-triplets
def correct_invalid_colors(my_list):


    return

# this fucntion removes the invalid color-triplets
def discard_invalid_colors(my_list):


    return
```

Figure: empty function definitions for the color correction task.

While writing your python program, you should consider using various list operations or methods (eg, *.append(), .remove(), .pop(), del*) as discussed during class. Empty python files have been uploaded, which you may find useful.

Topic

- Dictionary:
  - What is it?
  - Why do we need it?

Dictionary

- **List**: Access a position using **only numeric index**

```
student_id_list = [1002, 1003, 1004, 1005]
student_id_list[0]
student_id_list[1]
student_id_list[2]
student_id_list[3]
```

- **Dictionary**: Access an element using (eg number, string, character) as index (keys) to associate to something else (values)

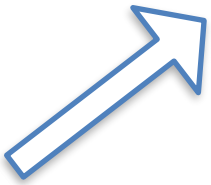
- Dictionary is an object that stores a collection of data
  - collection of key-value pairs
  - variable_name = { key_1 : $value_1$, key_2 : $value_2$, …, key_N : $value_N$ }

id	name
1002	Jack
1003	Daja
1004	Matt
1005	Simran

```
# dictionary with student ids as keys
students = {1002:"Jack", 1003:"Daja", 1004:"Matt", 1005:"Simran"}
```

Dictionary

- Dictionary is collection of key-value pairs or mapping between them
- variable_name = { key_1 : $value_1$, key_2 : $value_2$, ..., key_N : $value_N$ }
- The keys **must be unique**
  - Unlike previous example, keys below are **names**



name	id
Jack	1002
Daja	1003
Matt	1004
Simran	1005

```
# dictionary with student names as keys
students = {"Jack": 1002, "Daja":1003, "Matt":1004, "Simran":1005}
```

Retrieving a Value from a Dictionary

- Given *variable_name* = { key_1 : $value_1$, key_2 : $value_2$, …, key_N : $value_N$ }
- Use the syntax: *variable_name*[key]
- Since keys are **unique**, accessing via a key will return a specific value

```
# dictionary with student ids as keys
students = {1002:"Jack", 1003:"Daja", 1004:"Matt", 1005:"Simran"}
```

```
print("students[1002] --> ", students[1002])
print("students[1003] --> ", students[1003])
print("students[1004] --> ", students[1004])
print("students[1005] --> ", students[1005])
```

```
>>> %Run lec13.py

  students[1002] -->  Jack
  students[1003] -->  Daja
  students[1004] -->  Matt
  students[1005] -->  Simran
```

- Do you find similarity with anything else?

Creating a Dictionary

- There are several ways a dictionary can be created

- **Approach 1:** an empty dictionary

```
my_dict = {}
```

- **Approach 2:** with predefined entries

```
dict_student_scores = {'Reza':45, 'Chris':50, 'Sigi': 55}

dict_name_parts = {'Papa': 'John', 'Christiano':'Ronaldo', 'LeBron':'James'}

dict_random = {1:'one', (1,2):"two", None:"None keyword"}
```

Creating a Dictionary

- There are several ways a dictionary can be created

- **Approach 3:** with **dict** function with keyword args, unquoted-strings

```
my_dict = dict(Age=29, Tel=3405, Name='Kate')
```

- **Approach 4:** with **dict** function and list of two entries

```
my_dict = dict([[10, '10^1'], [100, '10^2'], [1000, '10^3']])
```