Python: Day 02

Data Structures

Previous Agenda

01

Introduction

What is Python?

02

Variables

Data Storage

03

Control Flow

Processing Information

04

Functions

Grouping Control Flows

05

Error Handling

Handling invalid code

06

Lab Session

Culminating Exercise

Agenda

01

Lists & Tuple

Ordered Group

02

Dictionary & Set

Unordered Group

03

String

Handling Text

04

File Handling

Data outside code

05

Comprehension

Iteration Shortcut

06

Lab Session

Culminating Exercise

01

List & Tuples

Ordered collection of items based on indices

List Definition

A list is a dynamic, ordered collection of items, defined using square brackets and commas

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  print(letters)
```

letters									
a	b	c	d	e	f	g	h	i	j

Make a new file and try making this list

List Looping

In general, for loops are used to iterate or go through groups of data

```
1  items = ['First Message', 'Second Message', 'Third Message']
2  for item in items:
3      print(item)
```

```
First Message
Second Message
Third Message
```

Quick Exercise: Colorful Printing

Make a list of colors and print them

```
colors = ...
print(colors)
```

Next, print each color in this format:

```
color: color 1
color: color 2
color: color 3
```

TIP: Make a new file for this

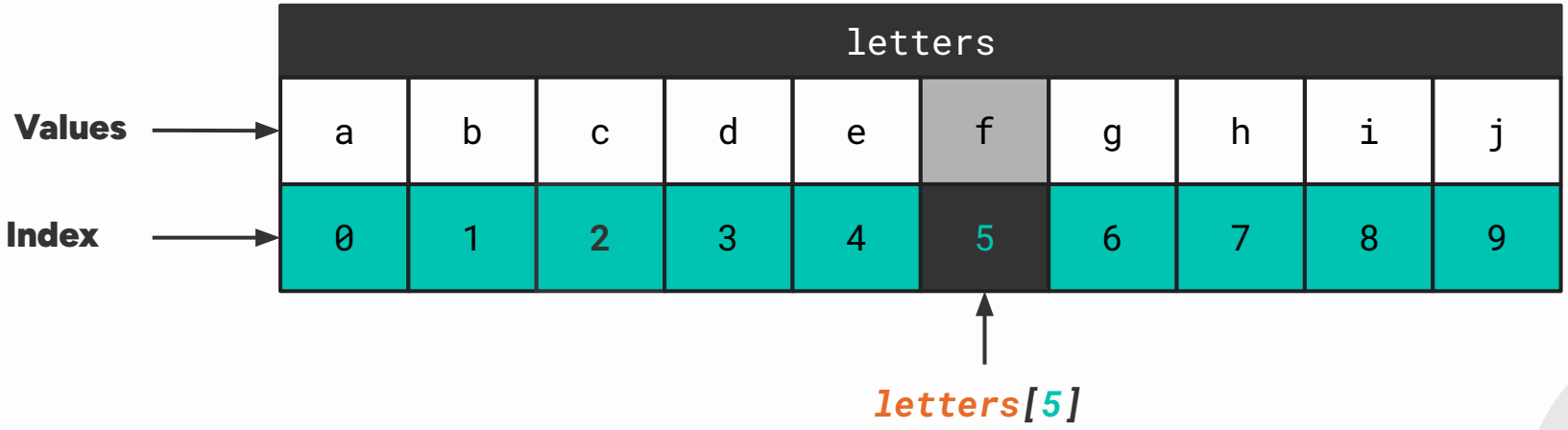
Index Logic

Always remember to start at zero

Item Access

Specific values can be accessed in a list by using the list name, square brackets, and index

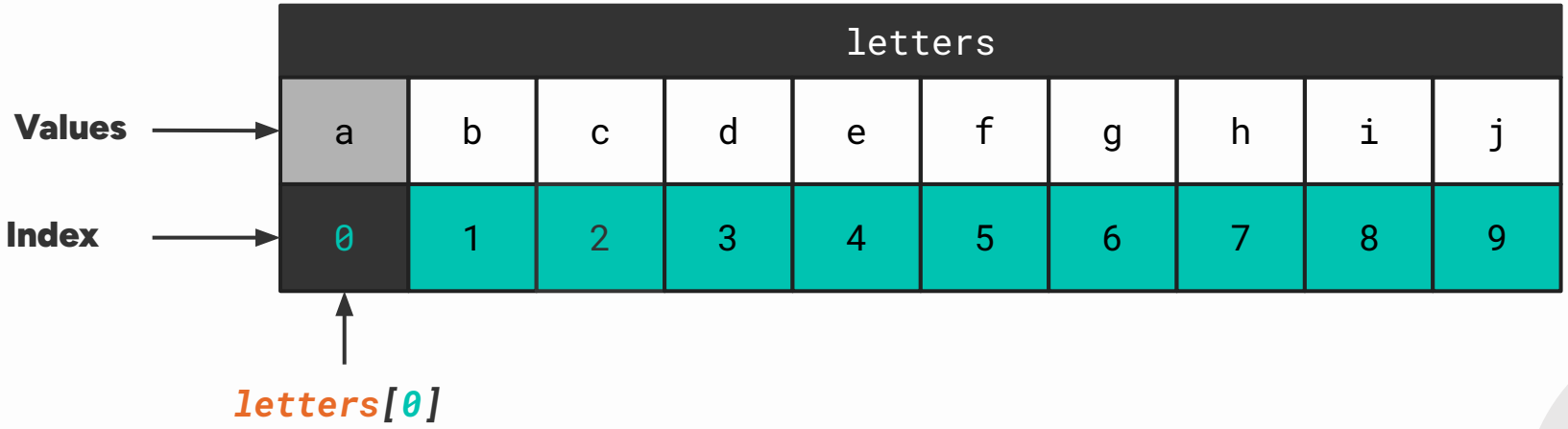
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[5])
```



Item Access

Specific values can be accessed in a list by using the list name, square brackets, and index.

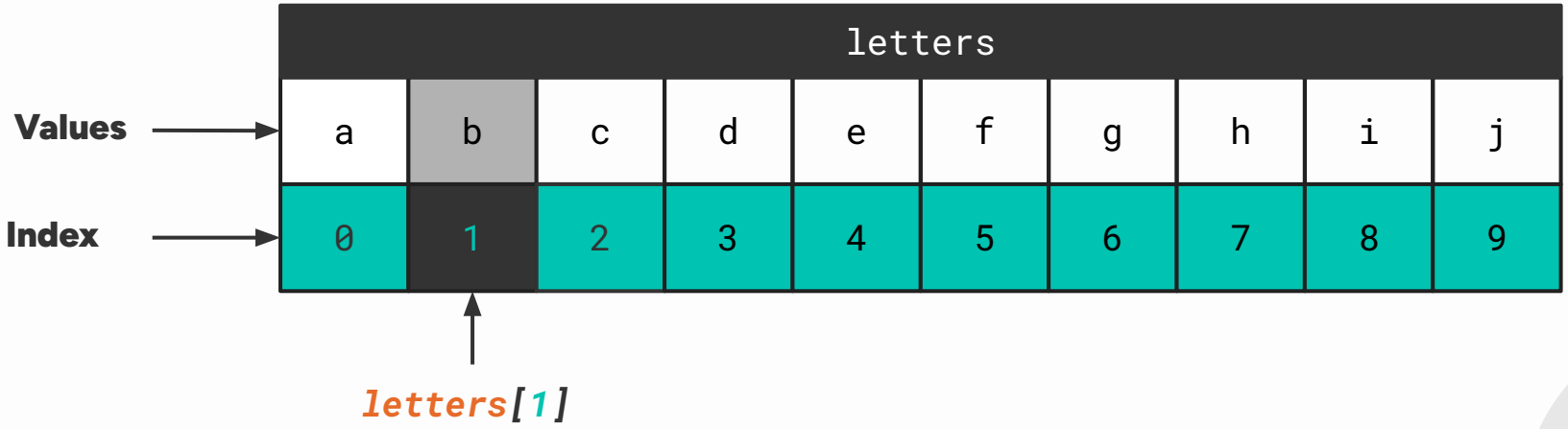
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[0])
```



Item Access

Specific values can be accessed in a list by using the list name, square brackets, and index.

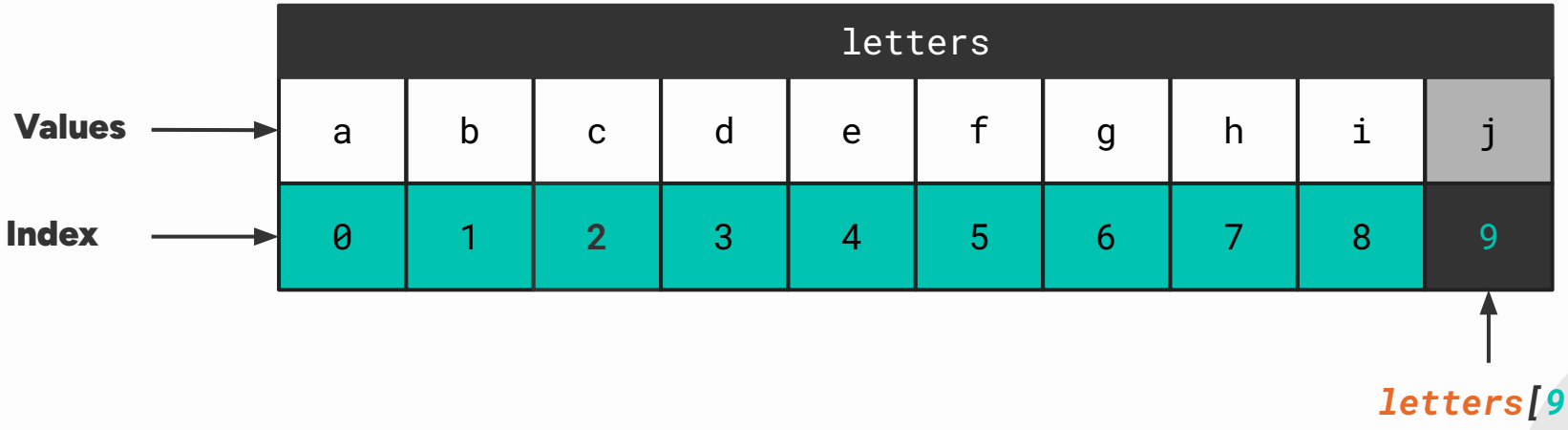
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[1])
```



Item Access

Specific values can be accessed in a list by using the list name, square brackets, and index.

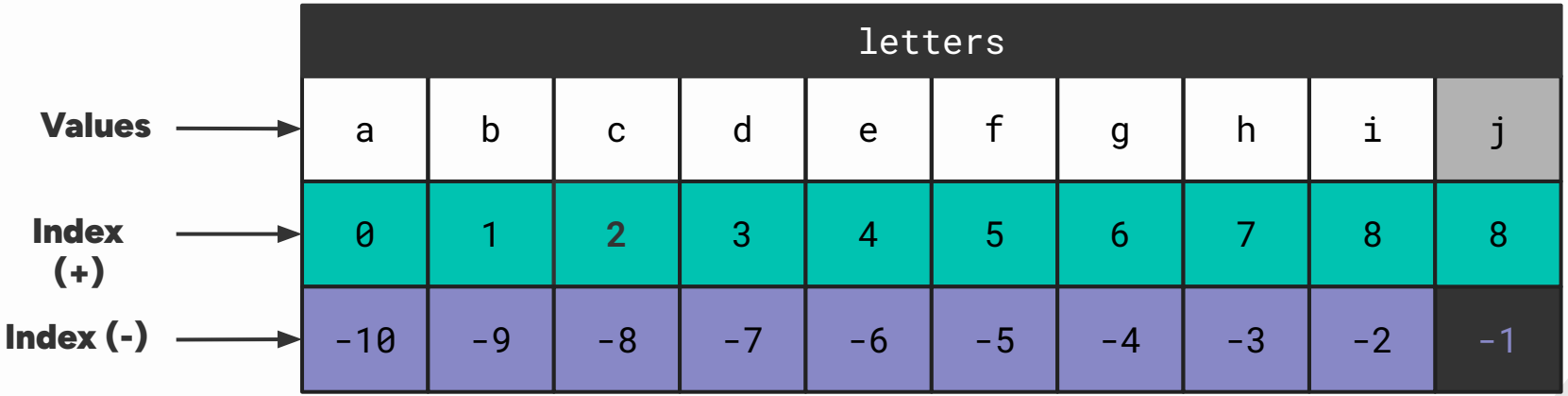
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[9])
```



Item Access

Specific values can be accessed in a list by using the list name, square brackets, and index.

```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[-1])
```



Item Access

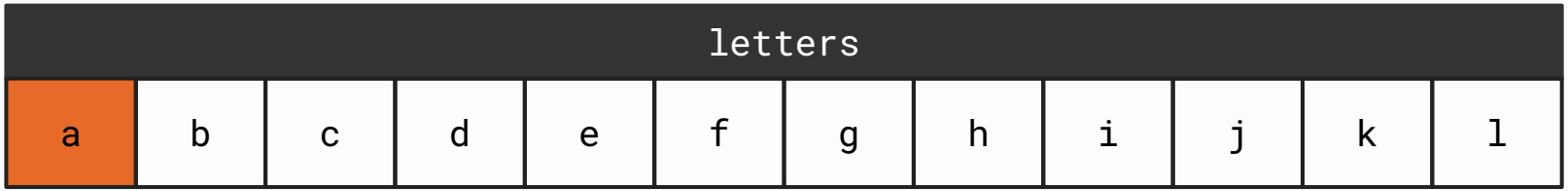
Specific values can be accessed in a list by using the list name, square brackets, and index.

```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[-2])
```

	letters									
Values →	a	b	c	d	e	f	g	h	i	j
Index (+) →	0	1	2	3	4	5	6	7	8	8
Index (-) →	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

ASAN NGA BA AKO SAYO?

Find the Index

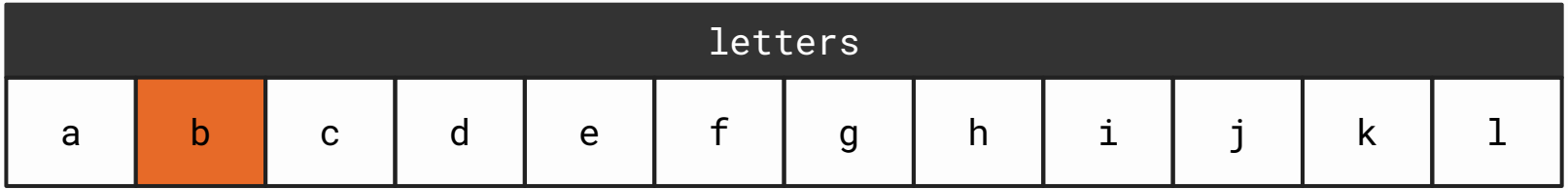


Find the Index

letters											
a	b	c	d	e	f	g	h	i	j	k	l
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

letters [0]
letters [-12]

Find the Index

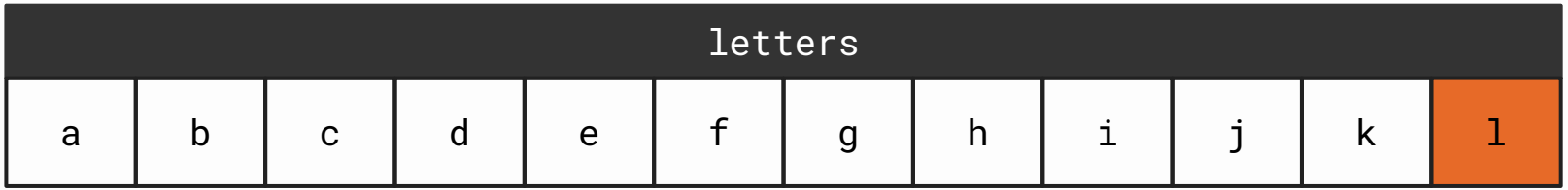


Find the Index

letters											
a	b	c	d	e	f	g	h	i	j	k	l
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

letters [1]
letters [-11]

Find the Index



Find the Index

letters											
a	b	c	d	e	f	g	h	i	j	k	l
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

letters [11]
letters [-1]

Find the Index

Find the Index

letters											
a	b	c	d	e	f	g	h	i	j	k	l
0	1	2	3	4	5	6	7	8	9	10	11
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

letters [6]
letters [-6]

Quick Exercise: Index Access

Given the following list:

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

Print the first, third, and fifth letter to create the following output

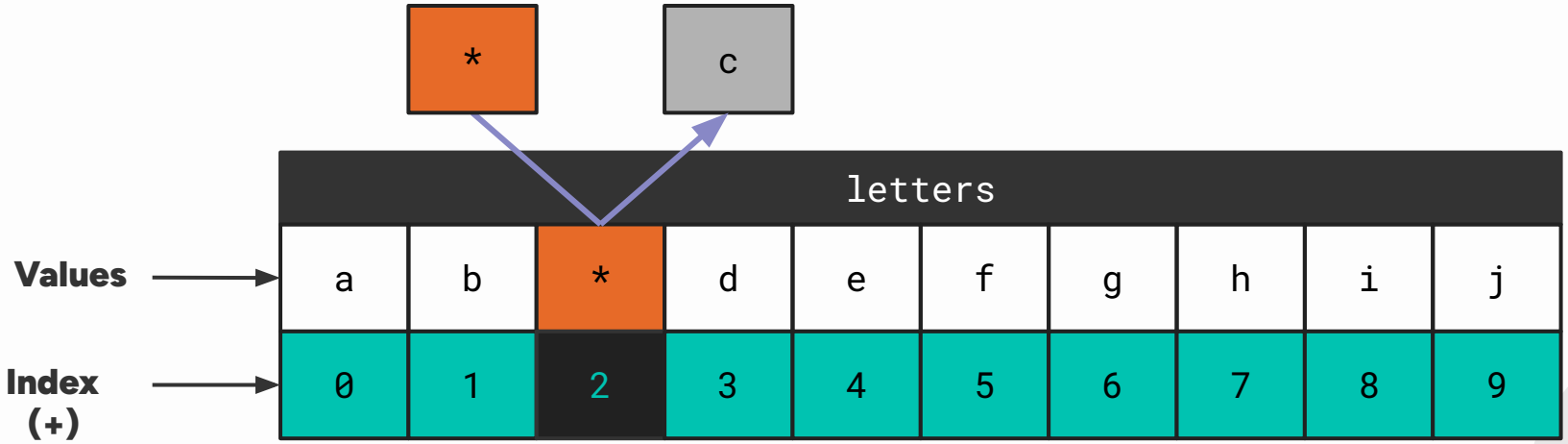
```
a
c
e
```

TIP: Make a new file for this

Item Modification

The item at a given index can be changed by accessing the index again like a variable

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[2] = '*'
```



Quick Exercise: Index Access

Given the following list:

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

Change the first, third, and fifth letter to an asterisk

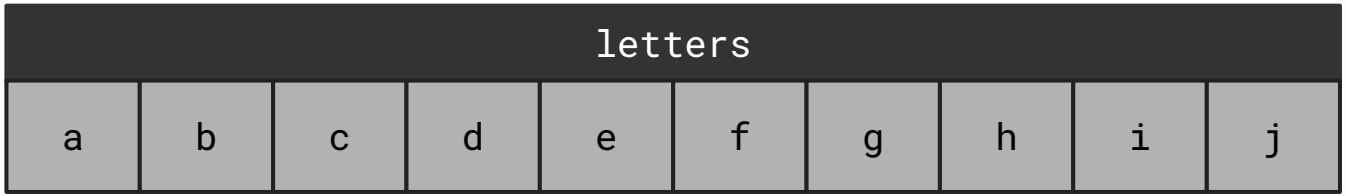
```
['*', 'b', '*', 'd', '*', 'f', 'g', 'h', 'i', 'j']
```

TIP: Make a new file for this

Tuple Definition

A tuple is a static, ordered collection of items, defined using parentheses and commas

```
1  letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
2  print(letters)
```

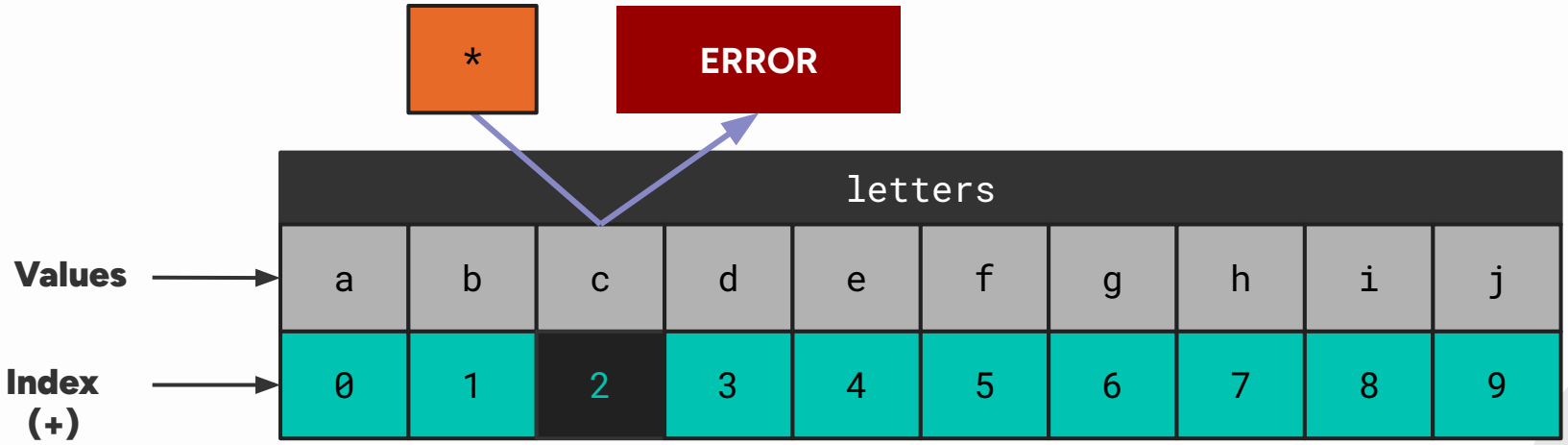


Make a new file and try making this tuple

Tuple Modification

Tuples cannot modify its contents after creation

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[2] = '*'
```



Conversion

Collections can change types like normal data types

List and Tuple Equality

A list and a tuple is not equal even if they have the same elements

```
example_list = [1, 2, 3]
```

≠

```
example_tuple = (1, 2, 3)
```

Two lists are not the same if they don't have the same elements in the same order

```
example_list1 = [1, 2, 3]
```

≠

```
example_list2 = [3, 2, 1]
```

Try the following

```
1 print([1, 2, 3] == (1, 2, 3))
2 print([1, 2, 3] == [3, 2, 1])
3 print((1, 2, 3) == (3, 2, 1))
```

List and Tuple Conversion

A list can be converted into a tuple and vice versa

```
1 print([1, 2, 3] == list((1, 2, 3)))
2 print(tuple([1, 2, 3]) == (1, 2, 3))
```

Try the given code in a new file

Nested Data

Real life data is often more complex

Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```

For this example, to access a specific value, you need to use indexing twice like this:

```
2  first_record = student_data[0]
3  first_record_score = first_record[1]
```

You can also directly access it by chaining indexing immediately

```
2  first_record_score = student_data[0][1]
```

Group Inside a Group

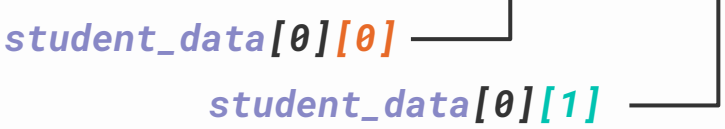
Lists and tuples can also contain lists or tuples inside them

```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```

Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

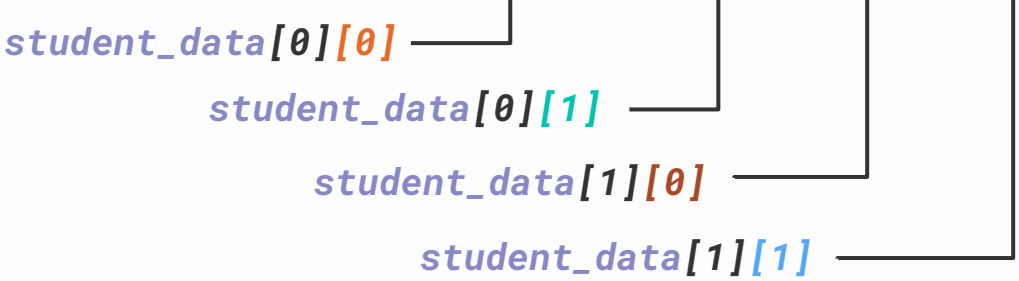
```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```



Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

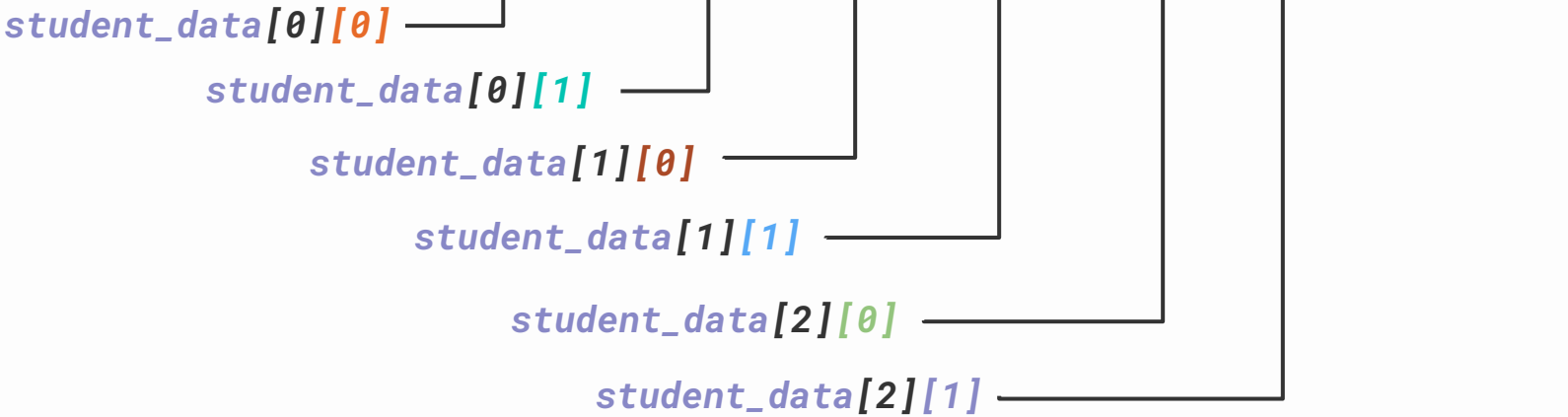
```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```



Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```

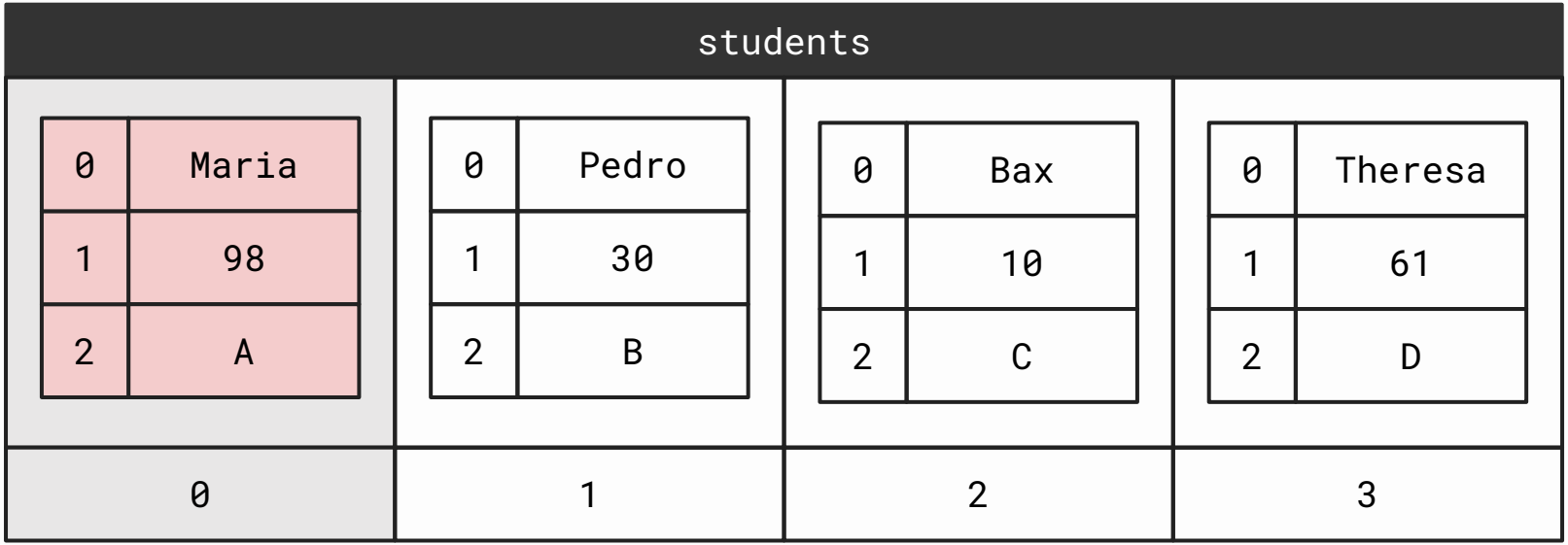



NASAAN KA NA
PolyEast RECORDS
LYRIC VIDEO

Find the Index

students			
0 | Maria 1 | 98 2 | A	0 | Pedro 1 | 30 2 | B	0 | Bax 1 | 10 2 | C	0 | Theresa 1 | 61 2 | D
0	1	2	3

Find the Index

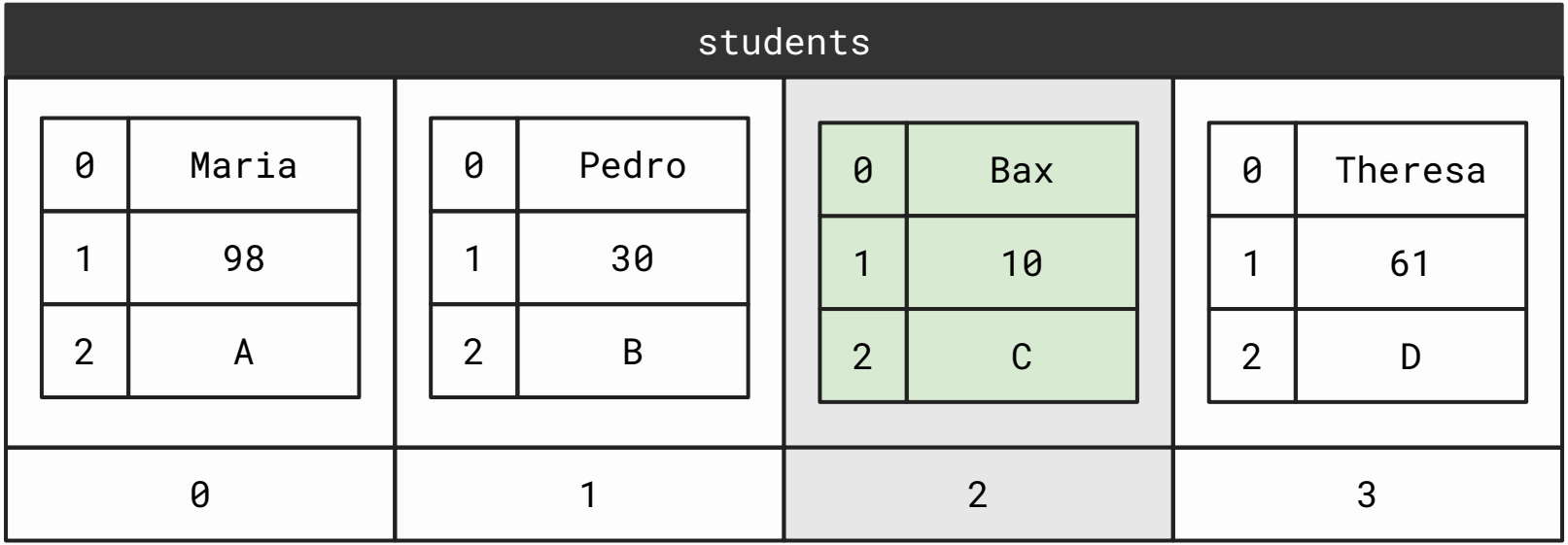


students [0]

Find the Index

students			
0 | Maria 1 | 98 2 | A	0 | Pedro 1 | 30 2 | B	0 | Bax 1 | 10 2 | C	0 | Theresa 1 | 61 2 | D
0	1	2	3

Find the Index



students [2]

Find the Index

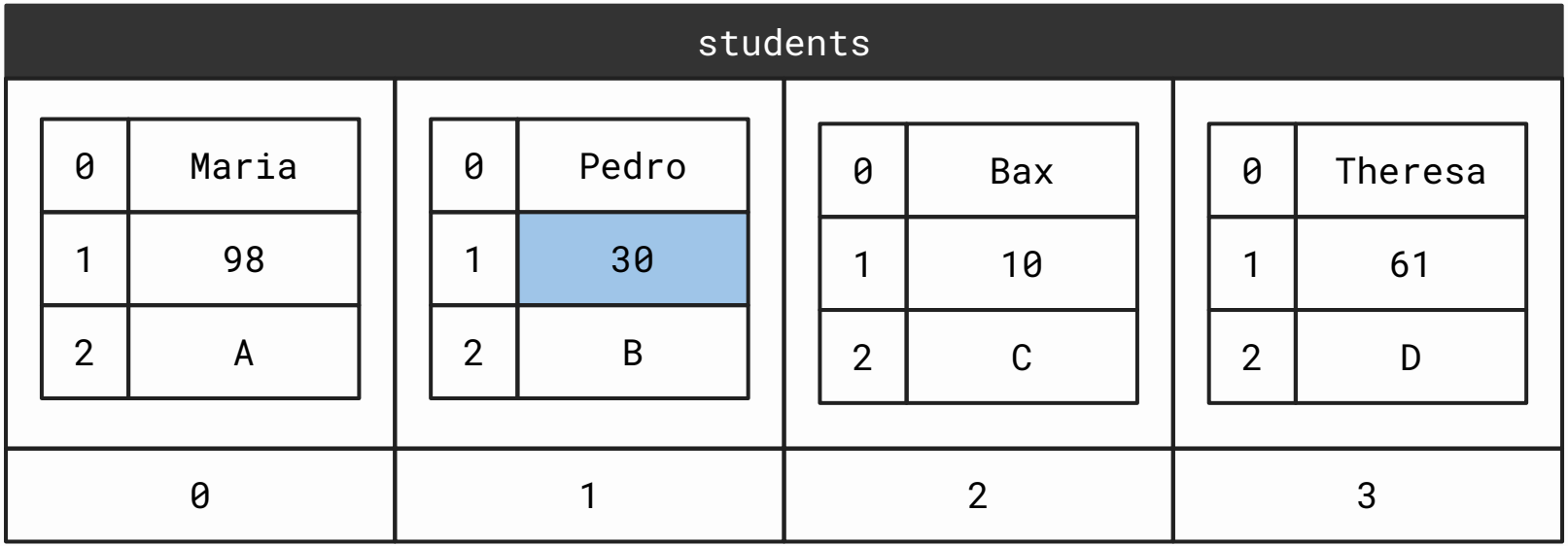
students			
0 | Maria 1 | 98 2 | A	0 | Pedro 1 | 30 2 | B	0 | Bax 1 | 10 2 | C	0 | Theresa 1 | 61 2 | D
0	1	2	3

Find the Index

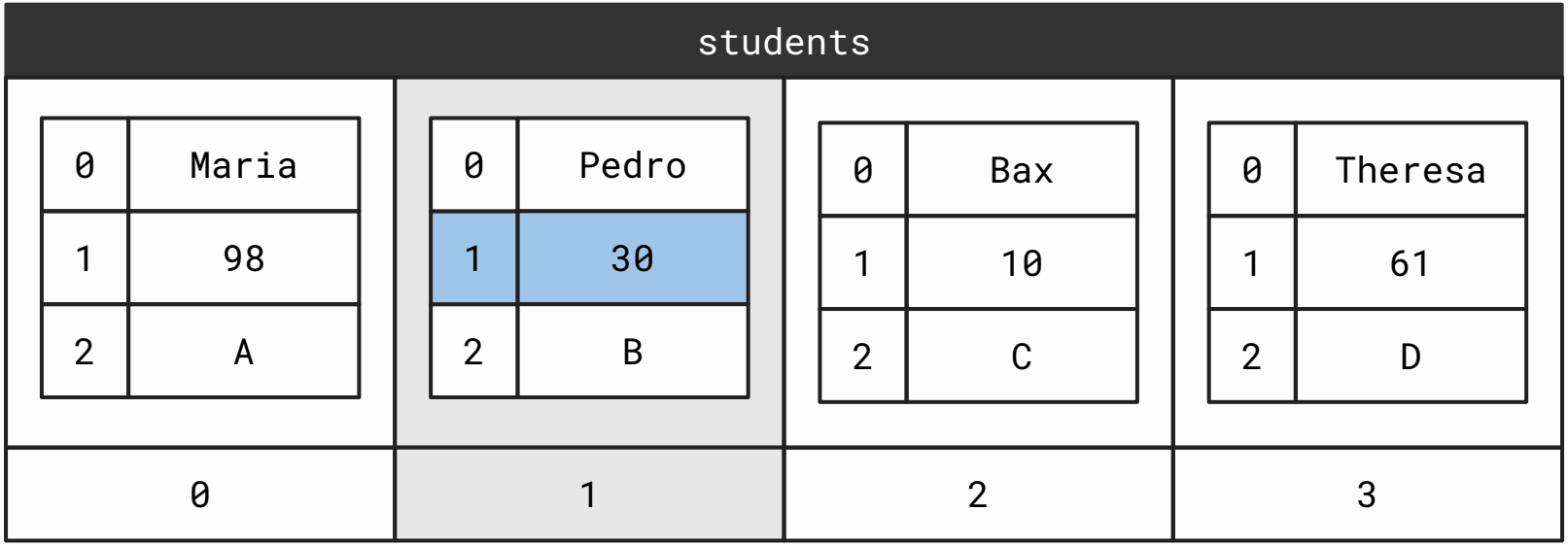
students			
0 | Maria 1 | 98 2 | A	0 | Pedro 1 | 30 2 | B	0 | Bax 1 | 10 2 | C	0 | Theresa 1 | 61 2 | D
0	1	2	3

students [3][1]

Find the Index



Find the Index



students [1][1]

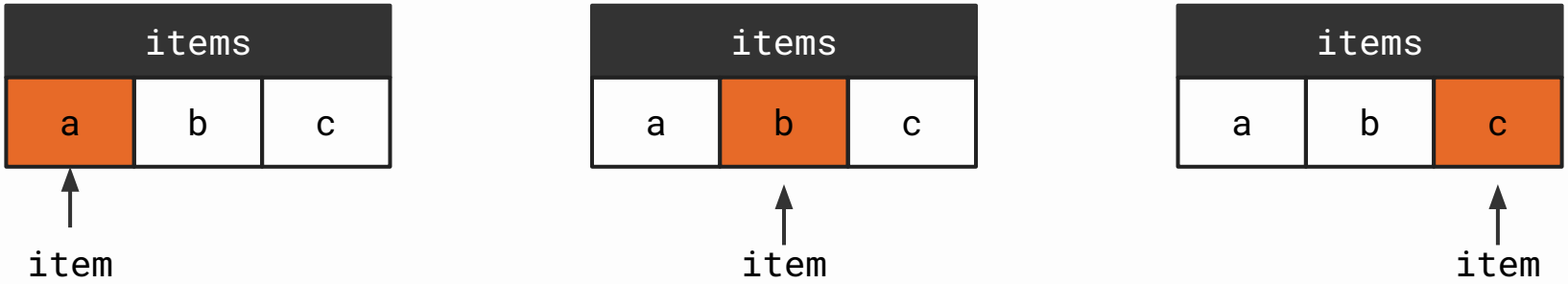
Loop Functions

Make looping more convenient

Default Looping

For loops are used to iterate or go through a sequence of items

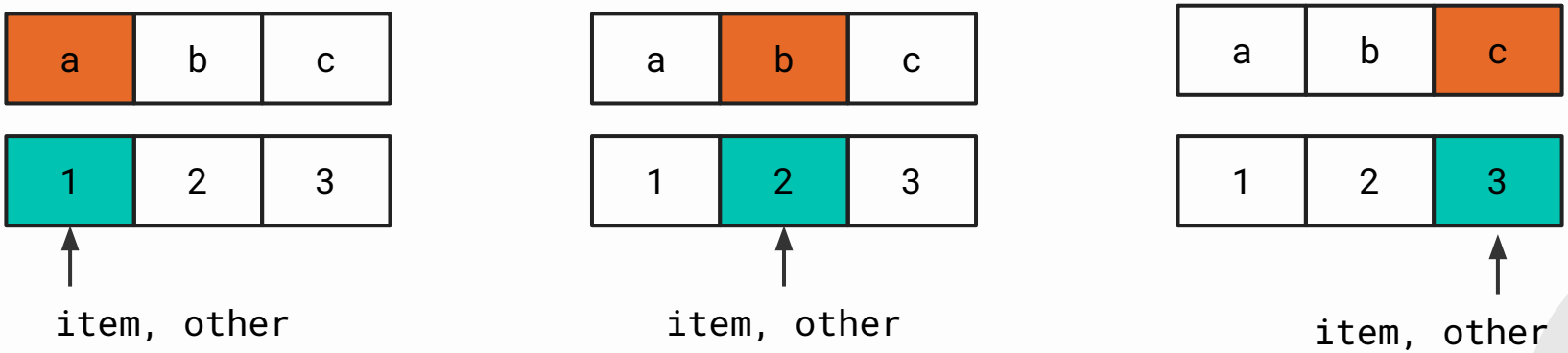
```
1  items = ['a', 'b', 'c']
2  for item in items:
3      print(item)
```



Multiple Looping

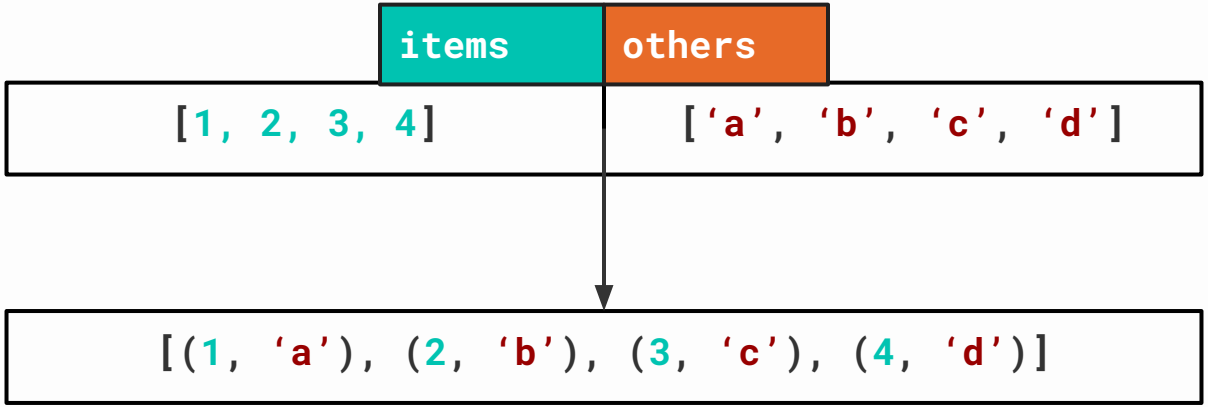
You can iterate through multiple items at once using the zip function

```
1  items = ['a', 'b', 'c']
2  others = [1, 2, 3]
3  for item, other in zip(items, others):
4      print(item, other)
```



Zipping

You can group the items of multiple lists or tuples together



Multiple Loopings Example

Here is another example of looping through multiple items at once.

```
names = ['Client 1', 'Client 2', 'Client 3']
balances = [10_000, 20_000, 3_000]
ids = [1, 2, 3]
```

```
for name, balance, id in zip(names, balances, ids):
    print(f"{id}: {name} ({balance} PHP)")
```

Make a new file and try this code

Quick Exercise: Student Records

Given the two lists

```
1  student_names = [“Juan”, “Maria”, “Joseph”]
2  student_scores = [70, 90, 81]
```

Print the student scores and names in the following format

```
Student Records:
    Record: Juan scored 70 in the exam.
    Record: Maria scored 90 in the exam.
    Record: Joseph scored 81 in the exam.
```

Make a new file for this exercise

Challenge: Print the highest scorer

Enumerate Looping

You can loop through a sequence of items and get the index using the enumerate function.

```
1  items = ['a', 'b', 'c']
2  for index, items in enumerate(items):
3      print(index, items)
```

```
0 a
1 b
2 c
```

Make a new file and try this code

Enumerate Looping (Different Start)

You can set the start of the enumerate function using the start parameter.

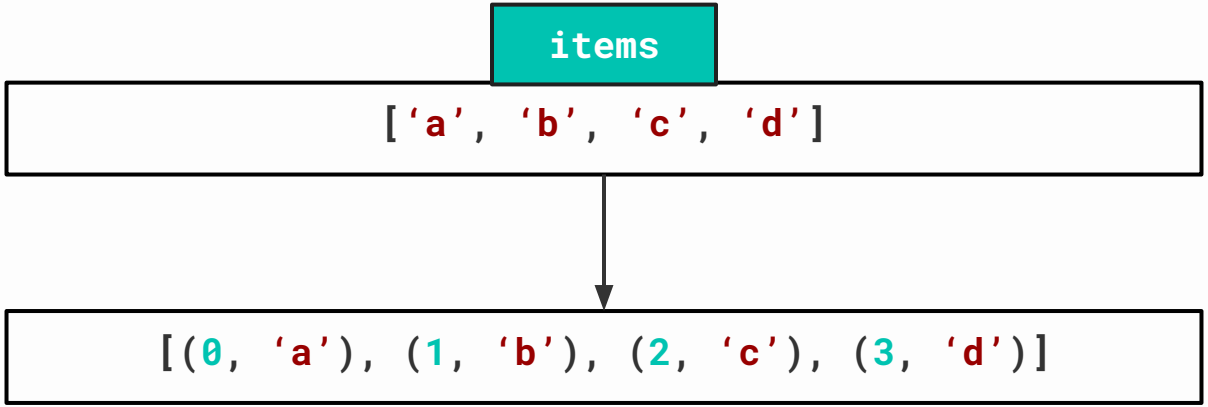
```
1 items = ['a', 'b', 'c']
2 for index, items in enumerate(items, start=1):
3     print(index, items)
```

```
1 a
2 b
3 c
```

Try and edit the previous code

Enumerate

A given list or tuple is paired to numbers from start to the last number



Quick Exercise: Attendance Log

Given the following list

```
1  student_names = [“Juan”, “Maria”, “Joseph”]
```

Print the student names in the following format

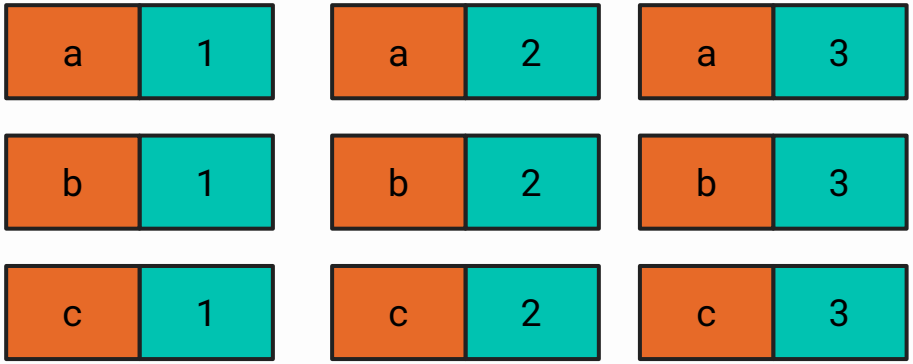
```
Attendance Log:
    Student 1: Juan
    Student 2: Maria
    Student 3: Joseph
```

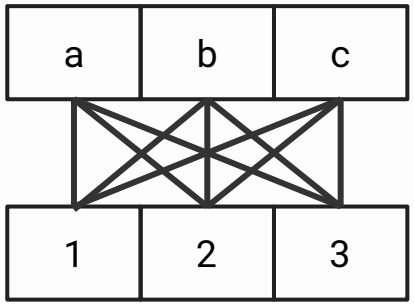
Make a new file for this exercise

Nested Looping

Using a loop inside another loop pairs every item to each other

```
1  items = ['a', 'b', 'c']
2  others = [1, 2, 3]
3  for item in items:
4      for other in others:
5          print(item, other)
```





Quick Exercise: Assignment

Given the two following lists, create every possible pairing of ***names*** and ***tasks***

```
1  names = [“Juan”, “Maria”, “Joseph”]
2  tasks = [“Task 1”, “Task 2”, “Task 3”]
```

```
Juan - Task 1
Juan - Task 2
Juan - Task 3
Maria - Task 1
Maria - Task 2
Maria - Task 3
Joseph - Task 1
Joseph - Task 2
Joseph - Task 3
```

Slicing

Using index logic to take more than one element

Slicing [Start:End]

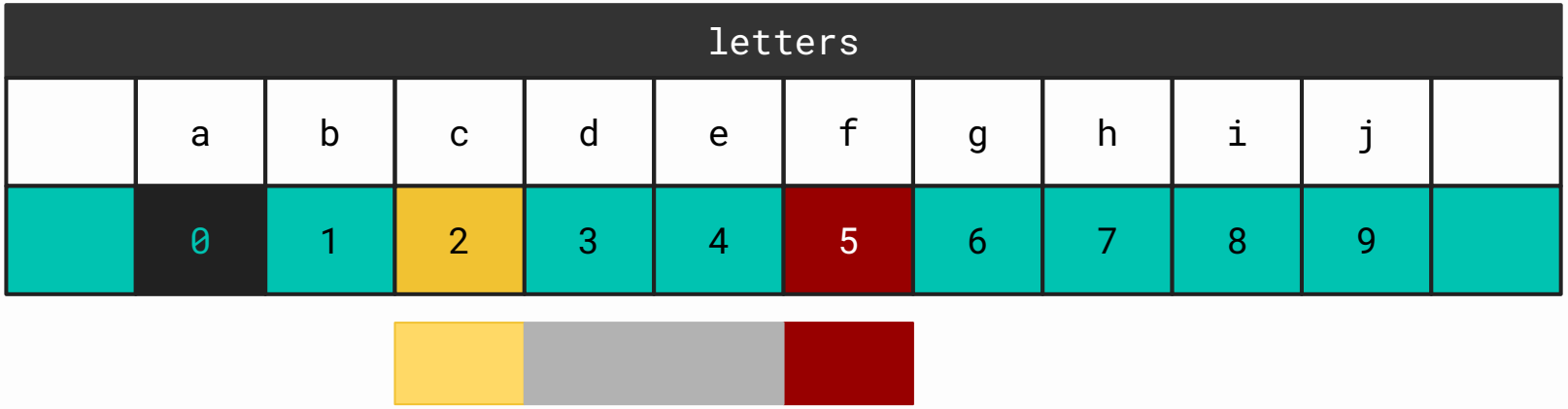
Lists and tuples can index multiple items as well using the slicing

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[start:end]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

Example 1

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[2:5]
```



['c', 'd', 'e']

Example 2

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[:4]
```

letters											
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	

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

Example 3

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[5:]
```

letters										
	a	b	c	d	e	f	g	h	i	j
	0	1	2	3	4	5	6	7	8	9
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

['f', 'g', 'h', 'i', 'j']

Example 4

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[-3:]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

['h', 'i', 'j']

Quick Exercise: Letter Slicing

Given the following list

```
1  letters = ['a', 'b', 'c', ..., 'y', 'z']
```

Print the following sublists

```
['a', 'b', 'c']
['x', 'y', 'z']
['h', 'i', 'j', 'k', 'l']
```

Make a new file for this exercise

Slicing [Start:End:Step]

Lists and tuples can index multiple items as well using slicing

```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 letters[start:end:step]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

Example 1

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[1:8:2]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

['b', 'd', 'f', 'h']

Example 2

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[::-1]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

['j', 'i', 'h', 'g', 'f', 'e', 'd', 'c', 'b', 'a']

Example 3

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  letters[::-2]
```

letters										
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

['j', 'h', 'f', 'd', 'b']

Quick Exercise: Letter Slicing

Given the following list

```
1  letters = ['a', 'b', 'c', ..., 'y', 'z']
```

Print the following sublists

```
['a', 'c', 'e', 'g']
['b', 'g', 'l', 'q']
```

Make a new file for this exercise

Containment

No need to loop through the entire collection to find things

Containment

One common operation used for collections is the in operator

```
1 food = ["ice cream", "burger", "fries"]
2 has_ice_cream = "ice cream" in food
3 print(has_ice_cream)
```

Conversely, you can check if an item is NOT in a data structure using the not in operator

```
1 food = ["ice cream", "burger", "fries"]
2 no_ice_cream = "ice cream" not in food
3 print(no_ice_cream)
```

Equality through Containment

One common use case for containment is to quickly check for equality

```
1  user_input = input("Proceed (Yes/yes/y)? ")
2  if user_input == "Yes" or user_input == "yes" or user_input =="y":
3      print("Proceeding")
```

This is an equivalent statement

```
1  user_input = input("Proceed (Yes/yes/y)? ")
2  if user_input in ("Yes","yes","y"):
3      print("Proceeding")
```

Quick Exercise: Allowed Lists

Create a list of names

```
names = [“Name 1”, “Name 2”, “Name 3”, ...]
```

Then ask the user for an input

```
user_name = input(“Please provide your name: “)
```

Then print the following depending on if the name is in `names`

```
Access Granted!
```

```
Access Denied!
```

TIP: Make a new file for this

Functions

Convenient functions for list and tuples

Min Function

Python has a min function that returns the smallest value in a given list or tuple

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 print(min(example))
3 print(example)
```

```
1
[1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

Make a new file and try a different input

Max Function

Python has a max function that returns the largest value in a given list or tuple

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 print(max(example))
3 print(example)
```

```
7
[1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

Make a new file and try a different input

Sum Function

Python has a sum function that returns the total of a list or tuple of numbers

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 print(sum(example))
3 print(example)
```

```
30
[1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

Make a new file and try a different input

Length Function

Python has a len function that returns the number of items in a list or tuple

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 print(len(example))
3 print(example)
```

```
10
[1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

Make a new file and try a different input

Quick Exercise: Class Statistics

Given the following scores

```
1  student_scores = [98, 75, 100, 86, 100, 3]
```

Add each in `student_scores`. Then, print the **lowest**, **highest**, and **average** score.

```
Lowest Score: 3
Highest Score: 100
Average Score: 77.0
```

TIP: Make a new file for this

Sorted Function (Ascending)

Python has a sorted function that returns a copy of the list or tuple in ascending order

```
1 example = [1, 3, 3, 5, 4]
```

```
2 print(sorted(example))
3 print(example)
```

```
[1, 3, 3, 4, 5]
[1, 3, 3, 5, 4]
```

Make a new file and try a different input

Sorted Function (Descending)

To create a sorted copy of a list or tuple, add a `reverse=True` in the sorted function

```
1  example = [1, 3, 3, 5, 4]
```

```
2  print(sorted(example, reverse=True))
3  print(example)
```

```
[5, 4, 3, 3, 1]
[1, 3, 3, 5, 4]
```

Make a new file and try a different input

Reversed Function

Python has a reversed function that returns a copy of the list, in reverse

```
1 example = [1, 3, 3, 5, 4]
```

```
2 print(reversed(example))
3 print(example)
```

```
[4, 5, 3, 3, 1]
```

Make a new file and try a different input

Quick Exercise: Student Rankings

From the previous exercise, print again the scores in ***student_scores***, but this time, in order of greatest to least.

```
Student Score 1: Highest score
Student Score 2: Second highest score
Student Score 3: Third highest score
...
```

Methods

Modifying the function directly

Recall: Functions can’t write outside

Functions can’t change variables outside because this is making another variable with the same name as the one outside

```
x = 10
def function():
    x = 5
    print(“Inner”, x)

print(“Outer”, x)
function()
print(“Outer”, x)
```

Make a new file and try this code

Recall: Functions can’t write outside

This mainly because the syntax for updating variables is unfortunately the same syntax for making a new one. In here, we’re making a new list with the same variable name

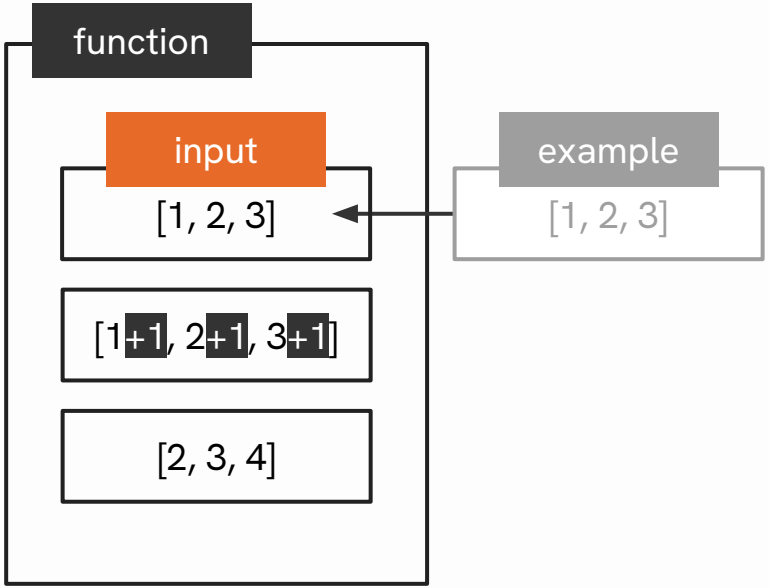
```
x = [1, 2, 3]
def function(x):
    x = [9, 8, 7]
    print(“Inner”, x)

print(“Outer”, x)
function()
print(“Outer”, x)
```

Make a new file and try this code

Functions

```
value = function(value)
```

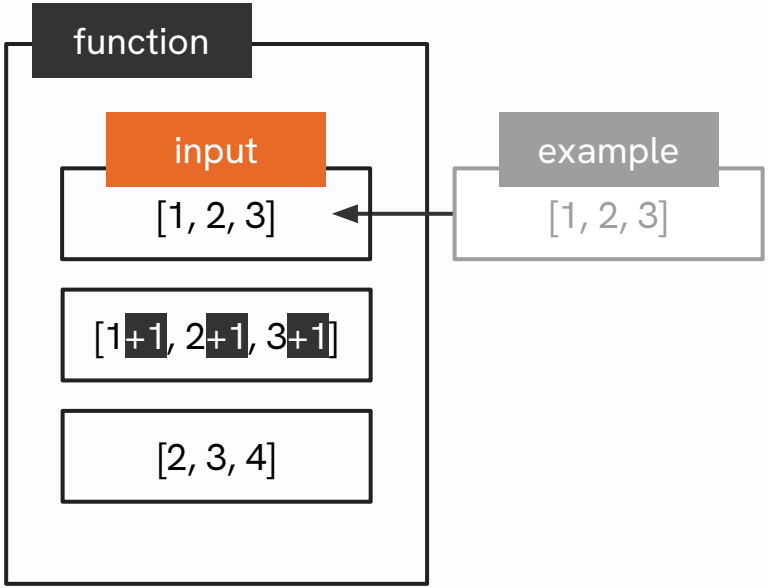


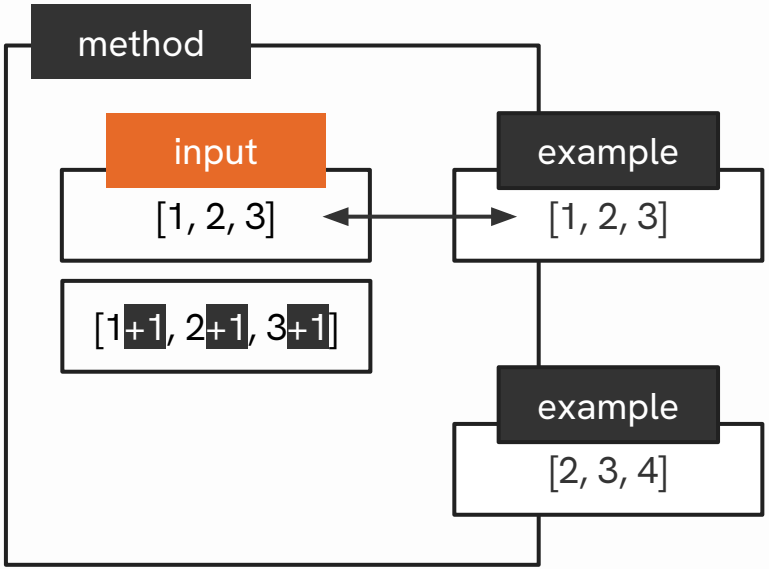
However, methods can!

Functions vs Methods

```
value = function(value)
```

```
value.method()
```





Methods Affects Readable Data

As long as the function can see the data, it can change it using methods

```
x = [1, 2, 3]
def function():
    x.append(999)

print(x)
function()
print(x)
```

Make a new file and try this code

Best Practice: Use Function Inputs

This is to make the intention clear that you will be changing a variable

```
x = [1, 2, 3]
def function(x):
    x.append(999)

print(x)
function()
print(x)
```

Make a new file and try this code

Append Method

A list has an append method that adds a new item to the end of the list

```
1  example = [1, 3, 3, 5, 4]
```

```
2  example.append(999)
3  print(example)
```

```
[1, 3, 3, 5, 4, 999]
```

Make a new file and try a different input

Extend Method

A list has an extend method that adds multiple items at the end of the list

```
1  example = [1, 3, 3, 5, 4]
2  other_example = [999, -1, 0]
```

```
3  example.extend(other_example)
4  print(example)
```

```
[1, 3, 3, 5, 4, 999, -1, 0]
```

Make a new file and try a different input

Insert Method

A list has an insert method that can add a value to before a specific index.

```
1 example = [1, 3, 3, 5, 4]
```

```
2 example.insert(0, 999)
3 print(example)
```

```
[999, 1, 3, 3, 5, 4]
```

Make a new file and try a different input

Quick Exercise: Basic Attendance

Given the empty list

```
1  attendee_names = []
```

Ask the user for an integer

```
2  attendee_count = int(input("How many attendees? "))
```

Based on the `attendee_count`, ask the user for that many attendee names

```
attendee name:
attendee name:
...
```

Append each input in `attendee_names` and print `attendee_names`

Index Method

A list or tuple has an index method that returns the index of a value. If it's not there, raises error

```
1 example = (1, 3, 3, 5, 4)
```

```
2 print(example.index(5))
```

```
3
```

Make a new file and try a different input

Count Method

A list or tuple has a count method that returns the number of instances of a given item

```
1 example = (1, 3, 3, 5, 4)
```

```
2 print(example.count(3))
```

```
2
```

Make a new file and try a different input

Quick Exercise: Name Matching

From the previous exercise, **print the number of attendees with the same name as you**.

```
Same Name: Number of names same as you
```

Remove Method

A list has an remove method that can remove a value from a list. Raises error if not there

```
1 example = [1, 3, 3, 5, 4]
```

```
2 example.remove(5)
3 print(example)
```

```
[1, 3, 3, 4]
```

Make a new file and try a different input

Safe Remove Method

It’s common to check if an item is in a list before removing it to avoid errors:

```
1  example = [1, 3, 3, 5, 4]
```

```
2  item_to_remove = 999
3  if item_to_remove in example:
4      example.remove(item_to_remove)
5  print(example)
```

```
[1, 3, 3, 4]
```

Clear Method

A list has a clear method that can remove all values

```
1 example = [1, 3, 3, 5, 4]
```

```
2 example.clear()
3 print(example)
```

```
[]
```

Make a new file and try a different input

Pop Method

The pop method removes a value for a given index

```
1 example = [1, 3, 3, 5, 4]
```

```
2 example.pop(-1)
3 print(example)
```

```
[1, 3, 3, 5]
```

Make a new file and try a different input

Pop Method with Return

If you want to know what value was removed, you can assign the method to a variable

```
1  example = [1, 3, 3, 5, 4]
```

```
2  removed_item = example.pop(-1)
3  print(removed_item)
4  print(example)
```

```
4
[1, 3, 3, 5]
```

Make a new file and try a different input

Quick Exercise: Late Attendee

From the previous exercise, **remove the last attendee name and print their name**

```
Late Attendee: Attendee Name
```

Sort (Ascending) Method

A list has a sort method that rearranges the items to ascending order

```
1  example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2  example.sort()
3  print(example)
```

```
[1, 1, 1, 1, 2, 3, 3, 5, 6, 7]
```

Make a new file and try a different input

Sort (Descending) Method

To sort the items in descending order, set `reverse=True`

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 example.sort(reverse=True)
3 print(example)
```

```
[7, 6, 5, 3, 3, 2, 1, 1, 1, 1]
```

Make a new file and try a different input

Reverse Method

A list has a reverse method that reverses the order of the current list

```
1 example = [1, 3, 3, 5, 6, 7, 1, 2, 1, 1]
```

```
2 example.reverse()
3 print(example)
```

```
[1, 1, 2, 1, 7, 6, 5, 3, 3, 1]
```

Make a new file and try a different input

Quick Exercise: Sorted Attendee

From the previous exercise, print again the names in `attendee_names`, but this time in **alphabetical order**

```
Attendee 1: Attendee Name
Attendee 2: Attendee Name
Attendee 3: Attendee Name
...
```

Public Playlist

Personal Playlist

Hands-on Exercise No. 1

Stephen • 142 songs, 10 hr 37 min

Custom order

#	Title	Album	Date added	
	Pwede Ba Lola Amour	Pwede Ba	2 weeks ago	5:43
2	dahan-dahan Music video • Lola Amour	dahan-dahan	2 weeks ago	4:24
3	Raining In Manila Lola Amour	Lola Amour	2 weeks ago	4:51
4	blue Music video • yung kai	blue	2 weeks ago	3:34
5	Abot Kamay Orange & Lemons	Strike Whilst The Iron Is Hot & Moonlane Gardens Collecti...	2 weeks ago	2:38
6	Weight of the World - English Version - J'Nique Nicole 岡部啓一	NieR:Automata Original Soundtrack	2 weeks ago	5:45
7	Weight of the World Kowaretasekainouta - Marina Kawano 岡部啓一	NieR:Automata Original Soundtrack	2 weeks ago	5:44
8	Get You (feat. Kali Uchis) Daniel Caesar, Kali Uchis	Freudian	2 weeks ago	4:38
9	Paruparo Sugarcane, JC Herrero	Paruparo	2 weeks ago	4:57

Personal Playlist

Pwede Ba
Lola Amour

Related music videos

Madali
Lola Amour, Al...

Please Don't...
Lola Amour

About the artist

Pwede Ba
Lola Amour

0:01 5:42

Personal Playlist

```
1  def add(song, playlist):
2      # Add song to playlist
3  def remove(song, playlist):
4      # Remove song from playlist
5  def play(playlist):
3      # Print the first song in the playlist (if any) and remove
4  def show_all(playlist):
5      # Print all contents in the playlist
6  def playlist_app():
7      # Ask user what command they want to do
8
9  playlist_app()
```

Challenge: Add Formatting

Challenge: Include Artist

Personal Playlist - Main Function

```
def playlist_app():
    # Initial Playlist (You can add starting songs)
    playlist = []

    # Let user select action to do
    user_choice = input("Select command: ")

    # Let user select action to do
    if user_choice == "add":
        new_song = input("Enter song name: ")
        add(playlist)
    elif ...
```

02

Dictionary & Set

Data focusing on relationships and mappings

Sets

Collection for unique record keeping

Set Definition

A set is a dynamic, unordered, unique collection of items

```
1  letters = {'a', 'a', 'b', 'c', 'd'}
2  print(letters)
```

letters
d, c, a, b

Make a new file and try making this set

Mutable Instances

Sets can only use non-mutable or static data types as values

Data Type	**Mutability**
`int, float, bool, None`	Not mutable (Static)
`string, tuple`	
`set`	Mutable (Dynamic)
`list`	
`dict`	

Set Add Method

Sets have a method add that takes an input value and adds it the set.

```
1  example = {1, 3, 5, 6}
```

```
2  print(example)
3  example.add(99)
4  print(example)
```

```
{1, 3, 5, 6}
{1, 99, 3, 5, 6}
```

Make a new file and try changing the input

Quick Exercise: Unique Attendance

Given the empty set

```
1  attendee_names = set()
```

Ask the user for an integer

```
2  attendee_count = int(input("How many attendees? "))
```

Based on the `attendee_count`, ask the user for that many attendee names

```
attendee name:
attendee name:
...
```

Append each input in `attendee_names` and print the unique `attendee_names`

Set Discard Method

Sets have a method discard that takes an input value and removes it (if it is in there)

```
1 example = {1, 3, 5, 6}
```

```
2 print(example)
3 example.discard(5)
4 print(example)
```

```
{1, 3, 5, 6}
{1, 3, 6}
```

Make a new file and try changing the input

Quick Exercise: Remove Special Case

From the previous exercise, remove the attendee with the same name as you

To verify that the last attendee name was removed, print `attendee_names` again

```
Attendee 1: Attendee Name
Attendee 2: Attendee Name
Attendee 3: Attendee Name
...
```

Set Pop Method

Sets have a method pop that randomly returns and removes a value in the set

```
1 example = {1, 3, 5, 6}
```

```
2 print(example)
3 return_value = example.pop()
4 print(example)
5 print(return_value)
```

```
{1, 3, 5, 6}
{3, 5, 6}
1
```

Make a new file and try changing the input

Quick Exercise: Raffle Winner

From the previous exercise, pick a random attendee and print their name

```
Raffle Winner: Attendee Name
```

Applicable Functions

Function Usage	Behavior
`len(example)`	Returns the number of items in a set
`min(example)`	Returns the lowest value in the set. Raises `ValueError()` if empty
`max(example)`	Returns the highest value in the set. Raises `ValueError()` if empty
`sum(example)`	Adds all items. Raises `TypeError()` if not numerical.
`sorted(example)`	Returns the sorted version of example (as a list)
`sorted(example,reverse=True)`	Returns the sorted version of example (as a list) (Descending order)

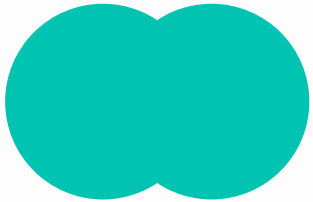
Set Operations

Operations specific to sets only

Set Union

```
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i', 'j'}
3  print(set1.union(set2))
4  print(set1 | set2)
```

Set1					
a	b	c	d	e	f

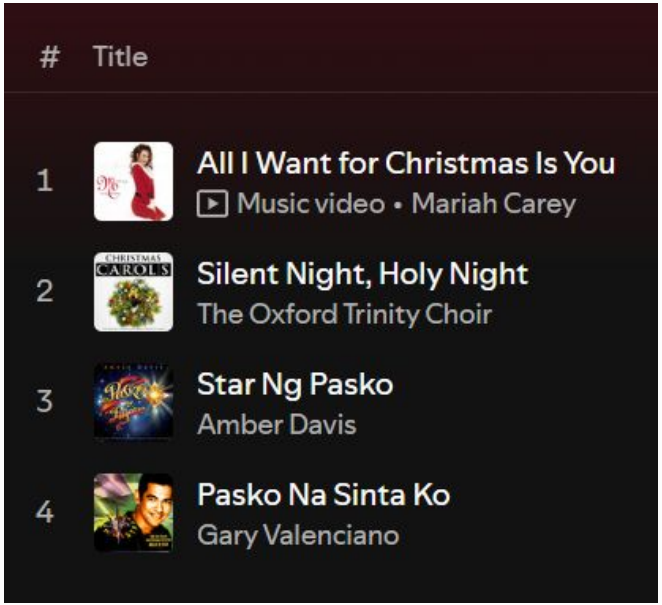


Set 2					
d	e	f	g	h	i

Quick Exercise: Combined Playlist

Create a new playlist that combines all the songs

Your Playlist

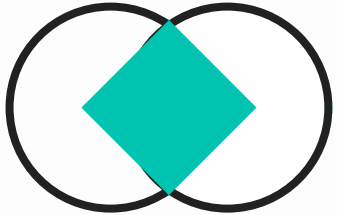


Friend Playlist

Set Intersection

```
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i', 'j'}
3  print(set1.intersection(set2))
4  print(set1 & set2)
```

Set1					
a	b	c	d	e	f



Set 2					
d	e	f	g	h	i

Quick Exercise: Mutual Playlist

Create a new playlist for songs that are in both playlist

Your Playlist

#	Title
1	 All I Want for Christmas Is You Music video • Mariah Carey
2	 Silent Night, Holy Night The Oxford Trinity Choir
3	 Star Ng Pasko Amber Davis
4	 Pasko Na Sinta Ko Gary Valenciano

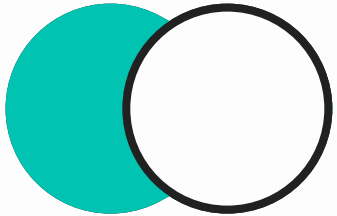
Friend Playlist

#	Title
1	 Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	 All I Want for Christmas Is You Music video • Mariah Carey
3	 Kampana Ng Simbahan Leo Valdez
4	 Christmas Bonus Aegis

Set Difference

```
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i', 'j'}
3  print(set1.difference(set2))
4  print(set1 - set2)
```

Set1					
a	b	c	d	e	f



Set 2					
d	e	f	g	h	i

Quick Exercise: Your Unique Songs

Create a new playlist for songs that only you have

Your Playlist

#	Title
1	 All I Want for Christmas Is You Music video • Mariah Carey
2	 Silent Night, Holy Night The Oxford Trinity Choir
3	 Star Ng Pasko Amber Davis
4	 Pasko Na Sinta Ko Gary Valenciano

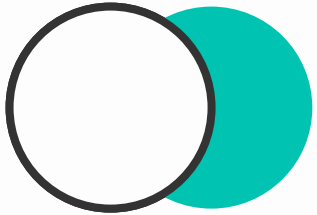
Friend Playlist

#	Title
1	 Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	 All I Want for Christmas Is You Music video • Mariah Carey
3	 Kampana Ng Simbahan Leo Valdez
4	 Christmas Bonus Aegis

Set Difference (Order Matters)

```
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i', 'j'}
3  print(set2.difference(set1))
4  print(set2 - set1)
```

Set1					
a	b	c	d	e	f

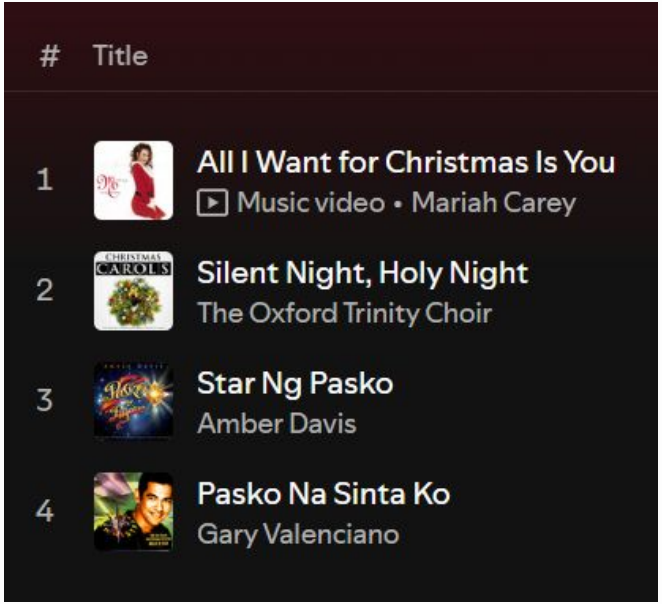


Set 2					
d	e	f	g	h	i

Quick Exercise: Friend Unique Songs

From the previous exercise, create a new playlist for songs that only your friend has

Your Playlist

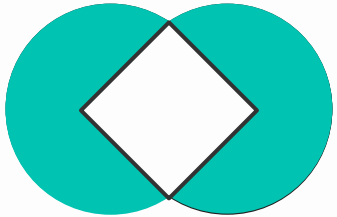


Friend Playlist

Set Symmetric Difference

```
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i', 'j'}
3  print(set1.symmetric_difference(set2))
4  print(set1 ^ set2)
```

Set1					
a	b	c	d	e	f



Set 2					
d	e	f	g	h	i

Quick Exercise: Unique Songs

Create a new playlist for songs that is not mutual

Your Playlist

#	Title
1	 All I Want for Christmas Is You Music video • Mariah Carey
2	 Silent Night, Holy Night The Oxford Trinity Choir
3	 Star Ng Pasko Amber Davis
4	 Pasko Na Sinta Ko Gary Valenciano

Friend Playlist

#	Title
1	 Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	 All I Want for Christmas Is You Music video • Mariah Carey
3	 Kampana Ng Simbahan Leo Valdez
4	 Christmas Bonus Aegis

Dictionary

The collection for convenient referencing

Student Scores and Names

student_scores			
70	98	81	80
0	1	2	3

student_names			
Juan	Maria	Joseph	Elise
0	1	2	3

Student Scores and Names (with Zip)

student_records			
(Juan, 70)	(Maria, 98)	(Joseph, 81)	(Elise, 80)
0	1	2	3

student_records [2][1]

“Joseph” ➔ 81

Student Scores and Names (Dict)

student_records			
70	98	81	80
Juan	Maria	Joseph	Elise

student_records ["Joseph"]

"Joseph" → 81

Student Name and Records

student_records							
70	A	98	B	81	C	80	D
Score	Group	Score	Group	Score	Group	Score	Group
Juan		Maria		Joseph		Elise	

student_records ["Joseph"]["Score"]

"Joseph" →"Score" → 81

Dictionary Definition

Dictionaries or dicts rely on the concept of a data called key providing access to a value. Similar to a regular key, there should only be one key to access a specific value.


```
1  student_records = {
2      “Juan”: 70,
3      “Maria”: 98,
4      “Joseph”: 81,
5      “Elise”: 80
6  }
```

Make a new file for the first one and try editing the list!

Example: Configuration

Consider the environment variables used by your app

```
config = {
    'FLASK_APP': 'my_flask_app',
    'FLASK_ENV': 'development',
    'DEBUG': True,
    'SECRET_KEY': 'your_secret_key_here',
    'DATABASE_URI': 'sqlite:///site.db',
    'CACHE_TYPE': 'simple',
    'SESSION_COOKIE_NAME': 'my_flask_session'
}
```

Quick Exercise: Favorites

Make a dictionary of your favorites (feel free to add more keys)

```
favorites = {
    “number”: 2,
    “color”: “black”,
}
```

Next, print your `favorites` dictionary

```
print(favorites)
```

TIP: Make a new file for this

Dictionary Access

The dictionary keys can be accessed using the same syntax as a list. If it’s not there,

```
1  student_records = {
2      “Juan”: 70,
3      “Maria”: 98,
4      “Joseph”: 81,
5      “Elise”: 80
6  }
7
8  print(student_records[“Joseph”])
```

```
81
```

Make a new file and try this code

Dictionary Access (Safe)

If you're not sure when a key is present, you can use the get method to return **None**

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7
8  print(student_records.get("Elizabeth"))
```

```
None
```

Edit the earlier code

Mapping

Dictionaries are often used to convert one value to another using key-value pairing

```
1  country_codes = {
2      "PH": "Philippines",
3      "US": "United States",
4  }
5
6  code = input("Enter country code: ")
7  print(f"{code} -> {country_codes[code]}")
8
```

Quick Exercise: Extend Codes

Add more country codes and test this code

```
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

code = input("Enter country code: ")
print(f"{code} -> {country_codes[code]}")

```

Make a new file for this

Dictionary Loops

Handling a set and list at once

Dictionary Iteration (Keys)

The dictionary keys can be accessed using the keys method

```
1  student_records = {
2      “Juan”: 70,
3      “Maria”: 98,
4      “Joseph”: 81,
5      “Elise”: 80
6  }
7
8  for student_name in student_records.keys():
9      print(student_name)
```

Make a new file and try this code

Dictionary Iteration (Keys)

The default for loop behavior of a dictionary is to return the keys

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7
8  for student_name in student_records:
9      print(student_name)
```

Make a new file and try this code

Quick Exercise: Key Iteration

Using your `favorites` dict earlier, print each key in this format:

```
My Favorites:
    Number
    Color
```

Dictionary Iteration (Values)

The dictionary values can be accessed using the values method

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7
8  for student_score in student_records.values():
9      print(student_score)
```

Make a new file and try this code

Quick Exercise: Value Iteration

Using your `favorites` dict earlier, print each value in this format:

```
My Favorites:
    2
    Black
```

Dictionary Iteration (Key-Value)

Both key and values can be accessed using the items method

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7
8  for student_name, student_score in student_records.items():
9      print(student_name, student_score)
```

Make a new file and try this code

Quick Exercise: Complete Iteration

Using your `favorites` dict earlier, print each key and value in this format:

```
My Favorites:
    Number: 2
    Color: Black
```

Dictionary Add

Dictionaries are write-safe at a cost

Dictionary Key Addition

To add a new entry, use the name of the dictionary, square brackets, and the key as well.

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7  student_records["Chocolate"] = 25
8  print(student_records["Chocolate"])
```

```
25
```

Quick Exercise: Modified Attendance

Given the empty dictionary

```
1  attendees = dict()
```

Ask the user for an integer

```
2  attendee_count = int(input("How many attendees? "))
```

Based on the `attendee_count`, ask the user for that many attendee names and task

```
attendee name:
attendee task:
...
```

Append each input in `attendees` and print `attendee_names`

Dictionary Overwriting

The same syntax for creating a new entry is used to modify an existing one

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7  student_records["Joseph"] = 100
8  print(student_records["Joseph"])
```

```
100
```

Dictionary Overwriting Guard

To avoid overwriting, double check if the key already exists using an if statement.

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7  if "Joseph" in student_records.keys():
8      print("Joseph is already recorded!")
9  else:
10     student_records["Joseph"] = 100
11 print(student_records["Joseph"])
```

```
81
```

Dictionary Common Methods

Here are some of the notable methods for dictionaries

Function Usage	Behavior
`my_dict.clear()`	Removes all entries in `my_dict`
`my_dict.pop(key)`	Remove and returns the entry with `key` in the `my_dict`. Will cause an error if it's not there.
`my_dict.update(other_dict)`	Merges two dictionaries (priority for `other_dict`)

Quick Exercise: Finished Attendee

Using your ***attendees*** dict earlier, ask the user for a name and remove that from ***attendees***

```
Attendee to Remove: Name of Attendee
```

Complex Data

Real-life data is often more challenging to handle

Single Entry

A dictionary can be thought of as a container for multiple related data

```
product_entry = {
    'name': 'Smartphone',
    'description': 'Latest model smartphone.',
    'price': 999.99,
    'stock': 25
}
```

Multiple Entries

By extension, you can make a list of those containers

```
product_catalog = [
    {
        'name': 'Smartphone',
        'description': 'Latest model smartphone.',
        'price': 999.99,
        'stock': 25
    },
    {
        'name': 'Wireless Headphones',
        'description': 'Noise-canceling wireless headphones.',
        'price': 199.99,
        'stock': 50
    },
]
```

Multiple Entries Iteration

When iterating a list of dictionaries, using the keys require manual placement in variables

```
15 for entry in product_catalog:
16     name = entry['name']
17     description = entry['description']
18     price = entry['price']
19     stock = entry['stock']
20 
21     print(f"{name} ({price}) [{stock}] - {description}")
```

Review: Single Entry

Make a dictionary of an employee (you can add more keys

```
employee = {
    'name': ‘John Doe’,
    'role': ‘Developer’
}
```

Next, place each value in a variable and print them

```
name = employee['name']
role = employee['role']
...
print(name)
print(role)
...
```

Quick Exercise: Multiple Entry

Make a list of dictionaries for employee details

```
employees = [
    {
        'name': ‘John Doe’,
        'role': ‘Developer’
        ...
    },
...
```

Then, for each employee entry in **employees**, print their information

```
for employee in employees:
    # Print details here
```

Complex Structure

Dictionaries can contain any data types, including lists and dictionaries

```
user_profile = {
    'name': 'Alice Smith',
    'preferences': {
        'language': ['English', ‘Japanese’],
        'notifications': True,
    }
}
```

Complex Structure

Dictionaries can contain any data types, including lists and dictionaries

```
1  user_profile = {
2      'name': 'Alice Smith',
3      'preferences': {
4          'language': ['English', ‘Japanese’],
5          'notifications': True,
6      }
7  }
```

user_profile [“preferences”]

{'language ': ['English' , ‘Japanese’], 'notifications' : True}

Complex Structure

Dictionaries can contain any data types, including lists and dictionaries

```
user_profile = {
    'name': 'Alice Smith',
    'preferences': {
        'language': ['English', ‘Japanese’],
        'notifications': True,
    }
}
```

user_profile [“preferences”]['language']

['English' , ‘Japanese’]

Complex Structure

Dictionaries can contain any data types, including lists and dictionaries

```
user_profile = {
    'name': 'Alice Smith',
    'preferences': {
        'language': ['English', ‘Japanese’],
        'notifications': True,
    }
}
```

user_profile [“preferences”]['language'][0]

'English'


HINAHANAP-HANAP KITA
vevo

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

student_report ["name"]

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

student_report ["activities"]

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

student_report ["activities"][1]

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

Example: Student Report

```
1  student_report = {
2      'name': 'Emma Green',
3      'grades': {'math': 85, 'science': 92, 'history': 78},
4      'activities': ['Debate Club', 'Soccer Team']
5  }
```

student_report ["grades"]["history"]

Example: Library System

```
library = {
    'branch': 'Central Library',
    'sections': [
        {
            'genre': 'Fiction',
            'books': [
                {'title': 'The Great Gatsby', 'copies': 5},
                {'title': '1984', 'copies': 2}
            ]
        }
    ]
}
```

Quick Exercise: Movie Entry

Create an example dictionary for all details required in a movie entry

```
movie_entry = {
    'title': 'Heneral Luna',
    'details': {
        'release_year': 2015,
        'genres': ['Historical', 'Drama', 'War'],
        'ratings': 90,
    },
    'cast': {
        'main_actors': ['John Arcilla', 'Mon Confiado'],
        'director': ['Jerrold Tarog'],
    },
}
```

H3

Cart System

Handle more than one type of information at a time

Cart System

```
1  def add(name, price, quantity, cart):
2      # Add a dictionary with key name to cart
3  def remove(name, cart):
4      # Remove entry with key name from cart
5  def show_all(cart):
3      # Print all contents in cart
4  def show_total(cart):
5      # Calculate and print total of cart
6  def cart_app():
7      # Ask user what command they want to do
8
9  cart_app()
```

Challenge: Add dictionary for discount code

03

Strings

Using extra functionalities for the most used data type

Special Strings

There are more forms to the standard string

Multiline String

If the string needs to span multiple lines, you can use a multiline string instead

```
1 message = """
2 Hello World
3 Hello World
4 Hello World
5 """
```

Result in Console:

```
Hello World
Hello World
Hello World
```

Docstrings

Adding a multiline string after a function definition serves as a guide called docstring

```
1  def helpful_function():
2      """
3      Adding a multiline string after a function definition
4      creates a guide when calling the help function
5      """
6      return 0
7
8  help(helpful_function)
```

Make a new file and try this code

Quick Exercise: Haiku Writing

Create a haiku and print it

```
haiku = """
    Crisp winds brush my face
    Golden leaves dance through the air
    Whispers of cold dawn
"""
print(haiku)
```

Make a new file for this exercise

F-String Formatting

F-strings also have the additional feature to add special formatting rules to its variables

```
f"Extra text {variable_name :codes}"
```

F-String: Decimal Places

F-strings can be used to limit the number of decimal places in a float variable

```
f"Extra text {number:.2f}"
```

Number of decimal places

Make a new file and try

```
1 number = 1.123456789
2 print(f"{number:.2f}")
```

Result in Console:

```
1.12
```

F-String: Commas

To add comma operations, you can just insert a comma before the dot

```
f"Extra text  {number:,}"
```

Number of decimal places with percentage

Make a new file and try

```
1  number = 123456789
2  print(f"{number:,}")
```

Result in Console:

```
123,456,789
```

F-String: Decimal Places with Commas

To add comma operations, you can just insert a comma before the dot

```
f"Extra text {number:,.2f}"
```

Number of decimal places with percentage

Make a new file and try

```
1 number = 123456.789
2 print(f"{number:,.2f}")
```

Result in Console:

```
123,456.79
```

F-String: Decimal with Percentage

F-strings can be used to change the float to percentage format

```
f"Extra text {number:.2%}"
```

Number of decimal places

Make a new file and try

```
1 number = 0.9899
2 print(f"{number:.2%}")
```

Result in Console:

```
98.99%
```

Quick Exercise: Simple Interest

Ask the user for the information of three items

Make a new file and try this

```
initial_balance = Input your initial balance
time_years = Input time elapsed (in years)
interest_rate = Input your the input rate
```

Then print the following output

```
Initial Balance: PHP Initial Balance (two decimal places)
Time (in Years): Years (four decimal places)
Interest Rate: Interest Rate in Percentage (four decimal places)
=====================================
Simple Interest: initial_balance * (1 + interest_rate) * time_years)
```

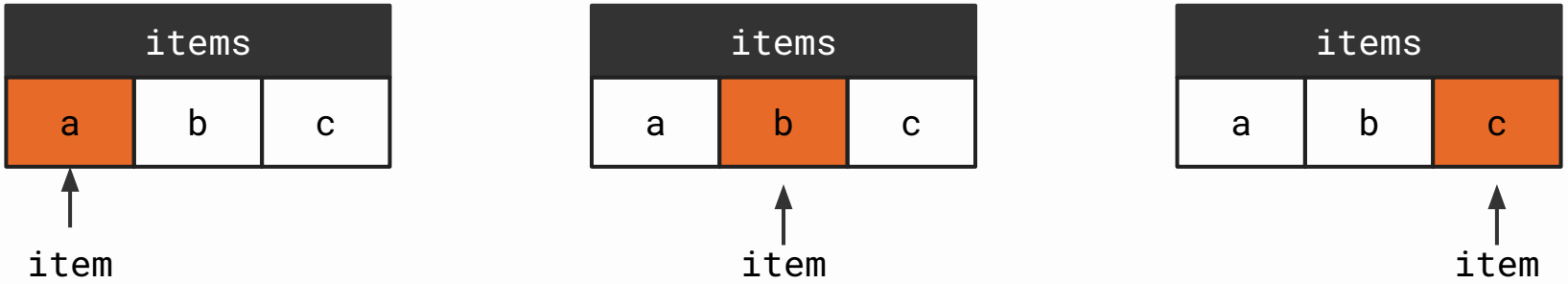
String as List

Remember that strings are a list of letters after all

String Looping

Using a for loop for a string will access the letters one at a time

```
items = 'abc'
for item in items:
    print(item)
```

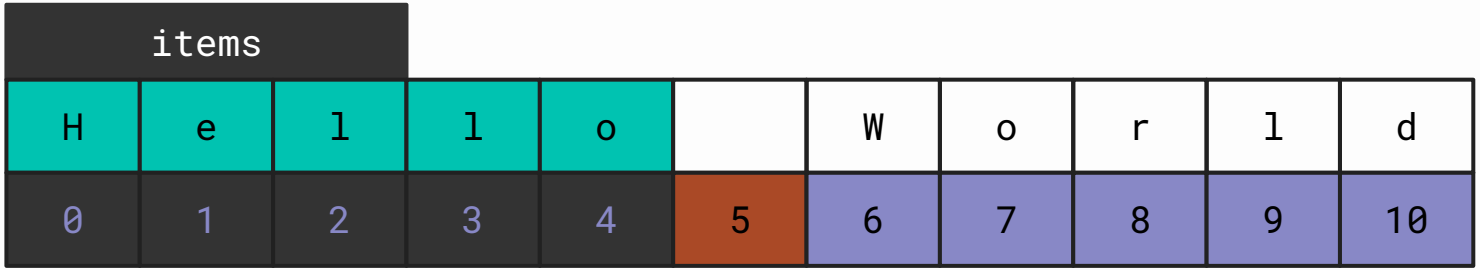


Make a new file and try this exercise

Substrings

Strings also support indexing and slicing access (not modification)

```
1  items = 'Hello World'
2  print(items[:5])
```



Make a new file and try this code

Substring Finding

Strings also support containment, but in a way that tries to find a substring instead.

```
1 message = 'Hello World'
2 print('World' in message)
```

```
True
```

Make a new file and try this code

Case Change

Applying formatting to an entire string

String Lowercase

Strings can be converted to lowercase using the lower() method.

```
1  example = “Hello World”
```

```
2  var_example = example.lower()
3  print(example)
4  print(var_example)
```

```
Hello World
hello world
```

Make a new file and try changing the input

String Uppercase

Strings can be converted to uppercase using the upper() method.

```
1  example = “Hello World”
```

```
2  var_example = example.upper()
3  print(example)
4  print(var_example)
```

```
Hello World
HELLO WORLD
```

Make a new file and try changing the input

String Title Case

Strings can be converted to title case using the title() method.

```
1  example = “This is a title”
```

```
2  var_example = example.title()
3  print(example)
4  print(var_example)
```

```
This is a title
This Is A Title
```

Make a new file and try changing the input

Use Case: Sanitized User Input

A very common use for the upper or lower method is to simplify the following code

```
1  user_input = input("Proceed (Yes/yes/y)? ")
2  if user_input == "Yes" or user_input == "yes":
3      print("Proceeding")
```

```
1  user_input = input("Proceed (Yes/yes/y)? ")
2  if user_input.lower() == "yes":
3      print("Proceeding")
```

Quick Exercise: Angery

Given a regular string input

```
I am perfectly calm and everything is fine
```

Convert to an angrier version

```
I AM PERFECTLY CALM AND EVERYTHING IS FINE
```

Case Check

Checking string formatting

String Check Lowercase

Strings have a method islower to return True if it's all lowercase. If not, returns False.

```
1  example = "hello"
```

```
2  all_lower = example.islower()
3  print(example)
4  print(all_lower)
```

```
hello
True
```

Make a new file and try changing the input

String Check Uppercase

Strings have a method isupper to return True if it's all uppercase. If not, returns False.

```
1  example = "HELLO"
```

```
2  all_upper = example.isupper()
3  print(example)
4  print(all_upper)
```

```
HELLO
True
```

Make a new file and try changing the input

String Check Space

Strings have a method isspace to return True if it's all space. If not, returns False.

```
1  example = "     "
```

```
2  all_space = example.isupper()
3  print(example)
4  print(all_space)
```

```
True
```

Make a new file and try changing the input

Quick Exercise: Case Closed

Given a regular string input

```
I am perfectly calm and everything is fine
```

Print the number of lowercase, uppercase, and spaces.

```
Lower case count: 34
Upper case count: 1
Space case count: 7
```

String Check Alphabet

Strings have a method isalpha to return True if it's all valid letters. If not, returns False.

```
1  example = "Hello"
```

```
2  all_alpha = example.isalpha()
3  print(example)
4  print(all_alpha)
```

```
Hello World
True
```

Make a new file and try changing the input

String Check Numeric

Strings have a method isnumeric to return True if it's all valid digits. If not, returns False.

```
1  example = "12345"
```

```
2  all_numeric = example.isnumeric()
3  print(example)
4  print(all_numeric )
```

```
12345
True
```

Make a new file and try changing the input

Quick Exercise: Number Check

Ask the user for an input

```
user_input = input(“Please provide any input: “)
```

Then print the following depending if the given input is a valid integer (without converting)

```
This is a valid number
```

```
This is not a valid number
```

TIP: Make a new file for this

String Edge

Check the start or end of a string

String Check Prefix

Strings have a method startswith() to return True if the string starts with its input.

```
1  example = “Hello World”
```

```
2  friendly = example.startswith(“Hello”)
3  print(example)
4  print(friendly)
```

```
Hello World
True
```

Make a new file and try changing the input

String Check Suffix

Strings have a method endswith() to return True if the string ends with its input.

```
1  example = "Hello World"
```

```
2  worldly = example.endswith("World")
3  print(example)
4  print(worldly)
```

```
Hello World
True
```

Make a new file and try changing the input

Quick Exercise: Email Check

Ask the user for an input

```
email_input = input(“Enter your email address: “)
```

Then print the following depending if the input is a valid gmail address.

```
This is a valid gmail.
```

```
This is not a valid gmail.
```

TIP: Make a new file for this

Word Handling

Common string methods to handle complex formatting issues

String Strip

Strings have a method strip() that returns the same string, but removes extra spaces on its ends

```
1  example = "          Hello World                    "
```

```
2  clean_example = example.strip()
3  print(example)
4  print(clean_example)
```

```
    Hello World
Hello World
```

Make a new file and try changing the input

Use Case: Sanitized User Input

A very common use for strip is to clean up extra spaces in user input

```
1  user_input = input("Proceed (Yes/yes/y)? ")
2  clean_input = user_input.lower().strip()
3  if clean_input == "yes":
4      print("Proceeding")
```

String Replace

Strings have a method replace() that returns the string but replaces a substring with another

```
1  example = "123,456,789"
```

```
2  alternative_example = example.replace(',', '.')
3  print(example)
4  print(alternative_example)
```

```
123,456,789
123.456.789
```

Make a new file and try changing the input

String Replace to Remove

The replace method can replace with an empty string to effectively remove the substring.

```
1  example = “a, b, c, d”
```

```
2  alternative_example = example.replace(“, “, ““)
3  print(example)
4  print(alternative_example)
```

```
a, b, c, d
abcd
```

Make a new file and try changing the input

Exercise: Number Cleanup

Given an erroneous positive integer that has spaces on the left and right, and uses commas

```
 2,444,111
```

Convert into a valid integer

```
2444111
```

Make a new file for this exercise

String Split

A string can be broken down into a list of substrings using the split method.

```
1  example = “Hello   I    am    a   message!”
```

```
2  words = example.split()
3  print(example)
4  print(words)
```

```
Hello   I    am   a   message!
['Hello', 'I', 'am', 'a', 'message!']
```

Make a new file and try changing the input

String Join

Conversely, a list of substrings can be combined using the join method.

```
1  example = ['Hello', 'I', 'am', 'a', 'message!']
```

```
2  combined_words = " ".join(example )
3  print(example)
4  print(combined_words)
```

```
['Hello', 'I', 'am', 'a', 'message!']
Hello I am a message!
```

Make a new file and try changing the input

Quick Exercise: Space Remover

Given a sentence with random extra spaces:

```
  This   is   an  excessively   spaced    sentence
```

Remove the extra spaces accordingly

```
This is an excessively spaced sentence
```

Make a new file for this

Regex

Non-linear way to handle string matching with exceptions

Regular Expressions

Regular expressions (regex or regexp) is a method for matching text based on patterns, defined using characters called **metacharacters**.

Metacharacter	Usage	Behavior
.	r"c.t"	Matches any single character except a newline.
*	r"a*bc"	Matches zero or more of the preceding character
+	r"a+bc"	Matches one or more of the preceding character
?	r"colou?r"	Matches zero or one of the preceding character
[ ]	r"[cb]at"	Matches one of the characters in square bracket
{n,m}	r"a{n,m}"	Matches preceding character from n to m times

Regular Expressions

Here is the syntax to handle more than one special character

Special Case	Behavior
[A-Z]	Matches a single uppercase letter
[a-z]	Matches a single lowercase letter
[A-Za-z]	Matches either a lowercase or uppercase letter
[0-9]	Matches a single digit
\w	Matches letters, digits, or underscores
\b	Matches a word boundary (start of the word)

Regex Find

A common use case for regex to find all instances of a given pattern within a larger text

```
1  import re
2
3  text = "Call me at 123-456-7890"
4  numbers = re.findall(r"\d+", text)
5  print(numbers)
```

Try this in a new file

Regex Replace

While Python strings already have the built-in replace method, the regex module also has a function for replacing substrings.

```
1  import re
2
3  text = "Alice has an apple and an avocado."
4  pattern = r"\ba\w*"
5  result = re.sub(pattern, "X", text)
6
7  print(result)
```

Try this in a new file

Simple Email Matcher

Consider the following pattern to match a simple email

```
> r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
```

email_text network suffix and locations

What could be the issue?

H4

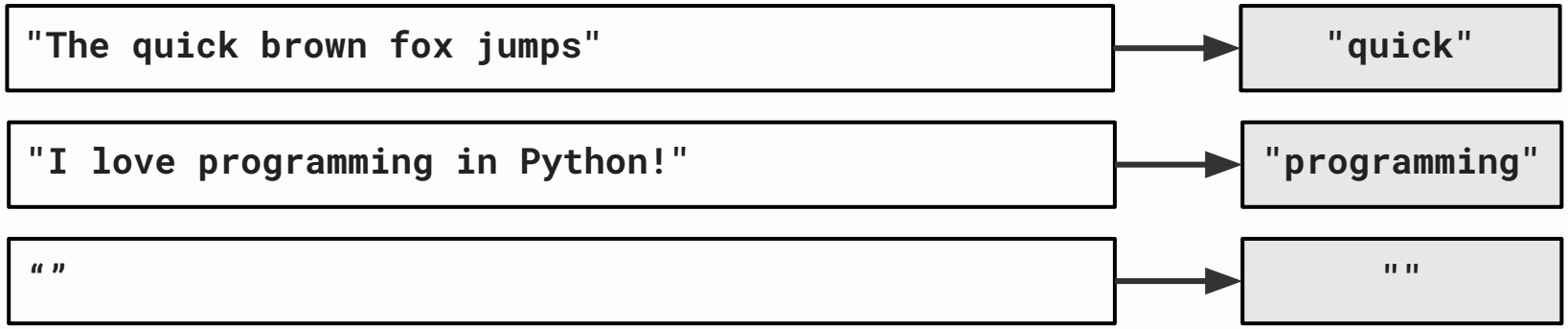
Longest Word

Pneumonoultramicroscopicsilicovolcanoconiosis

Longest Word

Make a function that takes an input text and returns the longest word (excluding special char)

```
def get_longest_word(text):
    # Add decoding process
    return longest_word
```



04

File Handling

More permanent approach to data

Text Files

The most common and well-known file type

File Operations

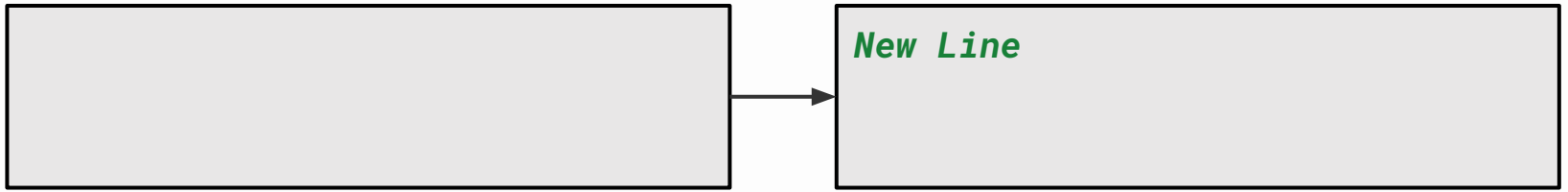
Python supports the following file operation modes to make persistent data (usually text)

File Modes	File Mode Name	When to Use
“r”	Read	For getting contents of file without writing
“w”	Write	For writing content into file from scratch
“a”	Append	For adding onto existing content into file from scratch

Writing Text File

A file can be managed by first using the **open()** function in the specified mode **“w”**. This returns a ***file*** that has the method ***file.write()*** to write contents

```
1  with open("test.txt", “w”) as file:
2      file.write(“New Line”)
```



Make a new file and try changing the input

Quick Exercise: Write Guestlist

The boss has given you a list of attendees that are allowed in the event. Write them in a file:

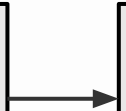
```
Mia Anderson
Ethan Roberts
Liam Johnson
Sophia Martinez
Olivia Davis
Noah Thompson
```

Appending Text File

A file can be managed by first using the **open()** function in the specified mode **“a”**. This returns a *file* that has the method *file.write()* to write contents below the current one

```
1  with open("test.txt", “a”) as file:
2      file.write(“\nNew Line”)
```

```
Current Line
```



```
Current Line
New Line
```

Make a new file and try changing the input

Quick Exercise: Append Guestlist

The boss forgot to add herself. Add her name in the last part

```
Mia Anderson
Ethan Roberts
Liam Johnson
Sophia Martinez
Olivia Davis
Noah Thompson
Alex Freze
```

Reading Text File (Full String)

A file can be managed by first using the **open()** function in the specified mode **“r”**. This returns a *file* that has the method *file.read()* to read contents

```
1  with open("test.txt", “r”) as file:
2      file_contents = file.read()
```

```
Existing Line 1
Existing Line 2
Existing Line 3
```

→ *file_contents*

Make a new file and try changing the input

Reading Text File (Line by Line)

A file can be managed by first using the **open()** function in the specified mode **“r”**. This returns a *file* that has the method *file.read()* to read contents.

```
1  with open("test.txt", “r”) as file:
2      file_contents = file.read().splitlines()
```

```
Existing Line 1
Existing Line 2
Existing Line 3
```

→ *file_contents*

Make a new file and try changing the input

Quick Exercise: Read Guestlist

Somebody wants to see the guest list in the terminal. Print out the guestlist in the terminal

```
Attendees:
    1.) Mia Anderson
    2.) Ethan Roberts
    3.) Liam Johnson
    4.) Sophia Martinez
    5.) Olivia Davis
    6.) Noah Thompson
    7.) Alex Freze
```

JSON

The text format of the internet

JSON File Format

JSON (JavaScript Object Notation) is a lightweight data format used for storing and transferring data. It represents data as key-value pairs and lists.

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john.doe@example.com",
  "is_active": true,
  "favorites": {
    "color": "blue",
    "food": "pizza"
  },
  "hobbies": ["reading", "cycling", "gaming"]
}
```

JSON Dump

Similar to csv file handling, json handling requires importing a library.

```
1  import json
2
3  data = [
4      {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
5      {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
6  ]
7
8  with open('people.json', 'w') as file:
9      json.dump(data, file)
```

Make a new file and try this code

JSON Dump (Formatted)

Similar to csv file handling, json handling requires importing a library.

```
1  import json
2
3  data = [
4      {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
5      {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
6  ]
7
8  with open('people.json', 'w') as file:
9      json.dump(data, file, indent=4)
```

Make a new file and try this code

Quick Exercise: Purchase Entries

Create a list of dictionaries containing dummy purchase information, then save it in a JSON file

```
purchase_entries = [
    {"Name": "Egg",   "Price": 10, "Description": "It's an egg."},
    {"Name": "Milk",  "Price": 50, "Description": "Fresh cow milk."},
    {"Name": "Bread", "Price": 35, "Description": "A whole loaf."},
    {"Name": "Apple", "Price": 30, "Description": "Fresh apple"},
    {"Name": "Rice",  "Price": 60, "Description": "A kilo of rice."},
]
```

JSON Load

Similar to csv file handling, json handling requires importing a library.

```
1  import json
2
3  with open('people.json', 'r') as file:
4      data = json.load(file)
5
6  print(data)
```

Make a new file and try this code

Quick Exercise: Load Purchases

Reload the entries saved in the previous JSON file and print it

```
Egg (Php 10): It's an egg.
Milk (Php 50): Fresh cow milk.
Bread (Php 35): A whole loaf.
Apple (Php 30): Fresh apple.
Rice (Php 60): A kilo of rice.
```

CSV Files

Handling table-like data that has rows and columns

CSV File Handling

Comma-Separated Values or CSV are simple version of the typical Excel or Google Sheet that only contains values, commonly separated by commas (sometimes by other char)

Name	Age	Occupation
Alice,	30,	Engineer
Bob,	25,	Designer
Charlie,	35,	Teacher

CSV Writing (with Lists)

```
1  import csv
2
3  data = [
4      ['Name', 'Age', 'Occupation'],
5      ['Alice', 30, 'Engineer'],
6      ['Bob', 25, 'Designer'],
7  ]
8
9  with open('people.csv', 'w', newline='') as file:
10     writer = csv.writer(file)
11     writer.writerows(data)
```

```
['Alice', 30, 'Engineer']
```

```
Alice, 30, Engineer
```

CSV Writing (with Dicts)

```
1  import csv
2
3  data = [
4      {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
5      {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
6  ]
7
8  with open('people.csv', 'w', newline='') as file:
9      writer = csv.DictWriter(file, fieldnames=data[0].keys())
10     writer.writeheader()
11     writer.writerows(data)
```

```
{'Name': 'Alice', 'Age': 30,
'Occupation': 'Engineer'}
```

```
Alice, 30, Engineer
```

Quick Exercise: Write the Attendee List

Create a csv with the given table:

Name	ID	Group
Alice	123	A
Bob	234	B
Charlie	999	C
Delta	441	D

CSV Reading (as Lists)

CSV Files can be read easily using a context manager and ***csv.reader(file)***.

```
1  import csv
2
3  with open('people.csv', 'r', newline='') as file:
4      reader = csv.reader(file)
5
6      for row in reader:
7          print(row)
```

CSV Reading (as Dicts)

CSV Files can be read easily using a context manager and ***csv.DictReader(file).***

```
1  import csv
2
3  with open('people.csv', 'r', newline='') as file:
4      reader = csv.DictReader(file)
5
6      for row in reader:
7          print(row)
```

Quick Exercise: Guard Check

Given a student name, ID, and Group, check if they are allowed.

```
Welcome to the school, please log:
    Name:
    ID:
    Group:
```

```
Hey! You're not on the list boss.
```

```
Please come in boss.
```

05

Comprehensions

Syntactic Sugar for creating data structures

List Comprehension

The most efficient way to generate a list of items

List Comprehension

List comprehensions are shortcuts to one of the most common process in Python

```
1
2
3
4
5
example_list = []
for number in range(11):
      example_list.append(number + 1)
```

```
1
example_list = [number + 1 for number in range(11)]
```

Make a new file and try this code

Quick Exercise: Number Generator

Ask the user for an integer

```
1  number_count = int(input("How many to generate? "))
```

Utilizing the concept of comprehensions, generate the following lists

List of Squares (x * x)

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, ...]
print(squares)
```

Skip by Five

```
fives = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, ...]
print(fives)
```

List Comprehension (with Conditions)

List comprehensions can also support conditions. If the condition isn't fulfilled, it isn't added.

```
example_list = []
for number in range(11):
    if number % 2 == 0:
        example_list.append(number)

```

```
example_list = [number for number in range(10) if number % 2 == 0]
```

Make a new file and try this code

Quick Exercise: Number Generator

Ask the user for an integer

```
1  number_count = int(input("How many to generate? "))
```

Utilizing the concept of comprehensions, generate the following lists

List of Odd Numbers

```
odds = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, ...]
print(odds)
```

Not Divisible by Five and Three

```
not_divisible_five_three = [1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, ...]
print(not_divisible_five_three)
```

Nested Data Creation

The most apparent use of list comprehensions is to immediately create data in specific formats

```
coordinates = [
    (x, y, z)
        for x in range(10)
        for y in range(10)
        for z in range(10)
]
print(coordinates)
```

Make a new file and try this code

You Control the Format

Using nested for loops doesn't mean you need to return a list or tuple

```
1  coordinates_sums = [
2      x + y + z
3          for x in range(10)
4          for y in range(10)
5          for z in range(10)
6  ]
7  print(coordinates_sums)
```

Make a new file and try this code

You Control the Format

Using nested for loops doesn't mean you need to return a list or tuple

```
1  coordinates_strings = [
2      f"({x}, {y}, {z})"
3          for x in range(10)
4          for y in range(10)
5          for z in range(10)
6  ]
7  print(coordinates_strings)
```

Make a new file and try this code

Data Filtering

Comprehensions can also be thought of as filters for what data to keep

```
1  requests = {“Andrew”: 10, “Peddy”: 21, “Alex”: 30}
2  banned = {“Alex”}
3
4  adults = [name for name, age in requests.items() if age >= 18]
5  print(adults)
6  allowed = [name for name in adults if name not in banned]
7  print(allowed)
```

Make a new file and try this code

Clean Comprehension Format

Comprehensions are recommended to be formatted in the following if they’re compex

```
1  def process(number):
2      return ((1 + number) // 2)** 3
3
4  def condition(number):
5      return number > 10
6
7  numbers = [991, 12, 89, 34, 121, 0]
8  data = [process(number) for number in numbers if condition(number)]
9  print(data)
```

Make a new file and try this code

Key Comprehension

Both sets and dictionaries can also use comprehensions

Set Comprehensions

Sets can also be created using comprehensions. They still keep unique values.

```
1  example_set = set()
2  text = "I am an igloo"
3  for char in text:
4      example_set.append(char)
5
```

```
1  text = "I am an igloo"
2  example_set = {char for char in text}
```

Make a new file and try the last code

Set Comprehensions (with Conditionals)

Sets can also be created using comprehensions. They still keep unique values.

```
1  example_set = set()
2  text = "I am an igloo"
3  for char in text:
4      if char != 'a'
5          example_set.append(char)
```

```
1  text = "I am an igloo"
2  example_set = {char for char in text if char != 'a'}
```

Make a new file and try the last code

Dict Comprehension

Finally, dictionaries can also be generated using comprehensions. They need key-values.

```
1  example_dict = dict()
2  for num in range(10):
3      example_dict[f"Key {num}"] = f"Value {num}"
4
5
```

```
1   example_dict = {
2       f"Key {num}": f"Value {num}" for num in range(3)
3   }
```

Make a new file and try the last code

Dict Comprehension (with Conditionals)

Additionally, dictionaries can also use conditions

```
1  example_dict = dict()
2  for num in range(10):
3      if num != 0:
4          example_dict[f"Key {num}"] = f"Value {num}"
5
```

```
1   example_dict = {
2       f"Key {num}": f"Value {num}" for num in range(3) if num != 0
3   }
```

Try editing the previous file with this

Nested Comprehension

Much like normal for loops, comprehensions can also be nested

```
numbers = {
    i: [x for x in range(1, i+1) if i % x == 0]
    for i in range(1, 6)
}

print(numbers)
```

Try this code in a new file

06

Lab Session

Defining and handling data


WORD
FIND

Initial Work: Word Bank

Create a dictionary called word_bank wherein the keys are the categories and the value is a list of words related to that category.

```
word_bank = {
    "Fruits": ["apple", "banana", "cherry", "mango"],
    "Animals": ["cat", "dog", "elephant", "lion"],
    "Countries": ["India", "Brazil", "France", "Japan"],
}
```

Game Setting: Word Selection

Ask the user for what category they want to pick (show the available categories).

```
Current Categories:
    1. Fruits
    2. Animals
    3. Countries

Choose a category: user input
```

Next, select a random word in the category (don't forget to import random)

```
possible_words = word_bank[user input]
word = choice(possible_words)
```

Actual Game: Letter Guessing

Show the selected word as underscores

```
_____
```

Ask the user for a letter input.

```
Enter letter: user input
```

If one of the letters of the word is in the selected word, reveal it

```
_pp__
```

While user has not guessed all the letter, keep asking for input.

Additions: Quality of Life Updates

Keep track of how many wrong guesses they made and reveal it once they guess all of the letters. Change your message depending on how many guesses they made

```
You guessed the word: apple
You made 0 incorrect guesses. That's amazing!
```

Prevent the user from entering a letter they already guessed before by asking again.

```
Enter letter: user input
That letter has already been guessed! Try again.
Enter letter:
```

Do the same process for selecting categories to prevent invalid categories.

Challenge: Dynamic Gameplay

At the start of the game, allow extra options for the user before game start

```
Current Categories:
    1. Fruits
    2. Animals

Options:
    1. Add category
    2. Add word
    3. Start Game
```

Make the game allow the user to play again after every end game.

```
You guessed the word: apple
You made 0 incorrect guesses. That's amazing!
Do you want to play again (y/n)? user_input
```

Prerequisite: Random Choice

In case we need to simulate randomness. First, put this at the top of your code.

```
1 from random import choice
```

This allows us to use the given function that returns a random item from a list

```
2 options = [“rock”, “paper”, “scissors”]
3 random_option = choice(options)
4 print(random_option )
```

Make a new file and try this code


Deck of Cards

Deck of Cards

```
def create_deck() -> list[str]:
    # Return a list of 52 strings containing a standard deck

def draw_top(deck: list[str], count: int=1)-> list[str]:
    # Remove count return count cards from the start from deck

def draw_bottom(deck: list[str], count: int=1) -> list[str]:
    # Remove and return count cards from the end of the deck

def draw_random(deck: list[str], count: int=1) -> list[str]:
    # Remove and return count random cards from the deck

def show(deck):
    # Print all cards in deck
```

Challenge: Dynamic Adding

```
def add_top(deck: list[str], other: list[str])-> list[str]):
    # Add cards in other to the first parts of deck

def add_bottom(deck: list[str], other: list[str])-> list[str]):
    # Add cards in other to the last parts of deck

def add_random(deck: list[str], other) -> list[str]):
    # Add cards in other randomly to deck
```

Final Challenge: Loading and Saving

```
def load(filename: str)-> list[str]):
    # Returns a list of cards loaded from a file

def save(deck: list[str], filename: str):
    # Saves a list of cards into a file (retrievable with load)
```

Sneak Peak

01

Definition

Data-Centric Approach

02

Hierarchy

Organizing Data

03

Polymorphism

Handling data types

04

Encapsulation

Data Hiding

05

GUI

Introduction to Tkinter

06

Lab Session

Culminating Exercise

Python: Day 02

Data Structures