Python: Day 02

Intermediate Python

Previous Agenda

01

Introduction

Understanding Python

02

Input-Output

Basic data processing

03

Control Flow

Processing Information

04

Functions

Basic Code Organization

05

Lab Session

Culminating Exercise

Current Agenda

01

Data Structures

Grouped data

02

Packaging

Handling Python Files

03

String+

Format & Methods

04

File Manage

Persistent Storage

05

Lab Session

Culminating Exercise

01

Data Structures

Predefined objects for grouping together related data

Sequences

Introduction ordered collection of items

List Definition

A list is a dynamic, ordered sequence of items

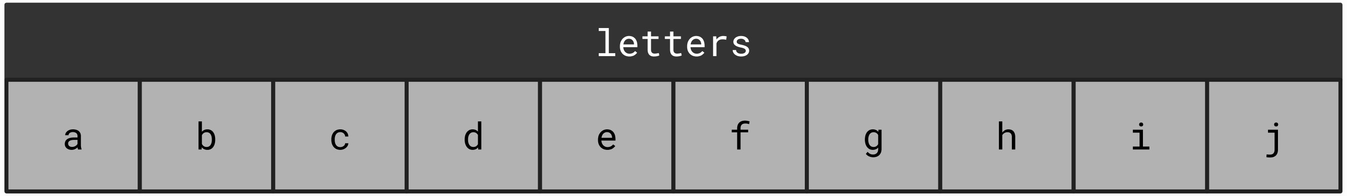
```
letters = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’]
print(letters)
```

letters									
a	b	c	d	e	f	g	h	i	j

Tuple Definition

A tuple is a static, ordered sequence of items

```
letters = (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’)
print(letters)
```



List and Tuple Equality

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

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

$\neq$

```
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]
```

$\neq$

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

List and Tuple Conversion

A list can be converted into a tuple and vice versa

```
list_example = [1, 2, 3, 4]
list_as_tuple = tuple(list_example)
```

```
tuple_example = (1, 2, 3, 4)
tuple_as_list = tuple(tuple_example)
```

Containment

The **in** operator is used to check if an item is included in the list

```
food = [“ice cream”, “burger”, “fries”]
has_ice_cream = “ice cream” in food
print(has_ice_cream)
```

Conversely, the **not in** operator is used to check if an item is NOT included in the list

```
food = [“ice cream”, “burger”, “fries”]
no_ice_cream = “ice cream” not in food
print(no_ice_cream)
```

Quick Exercise: Guest List

Create a tuple 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`

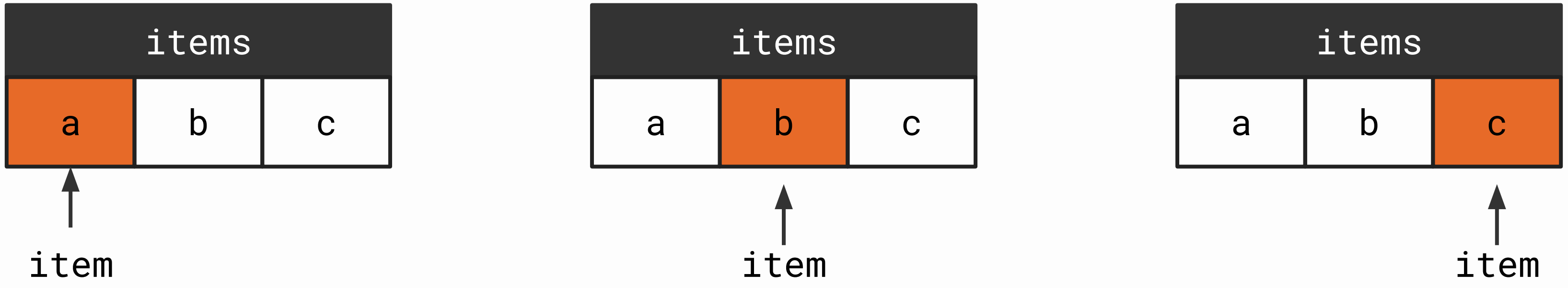
```
Access Granted!
```

```
Access Denied!
```

Default Loop

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

```
items = ('a', 'b', 'c')
for item in items:
    print(item)
```



Quick Exercise: Colorful Printing

Make a tuple of colors and print them

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

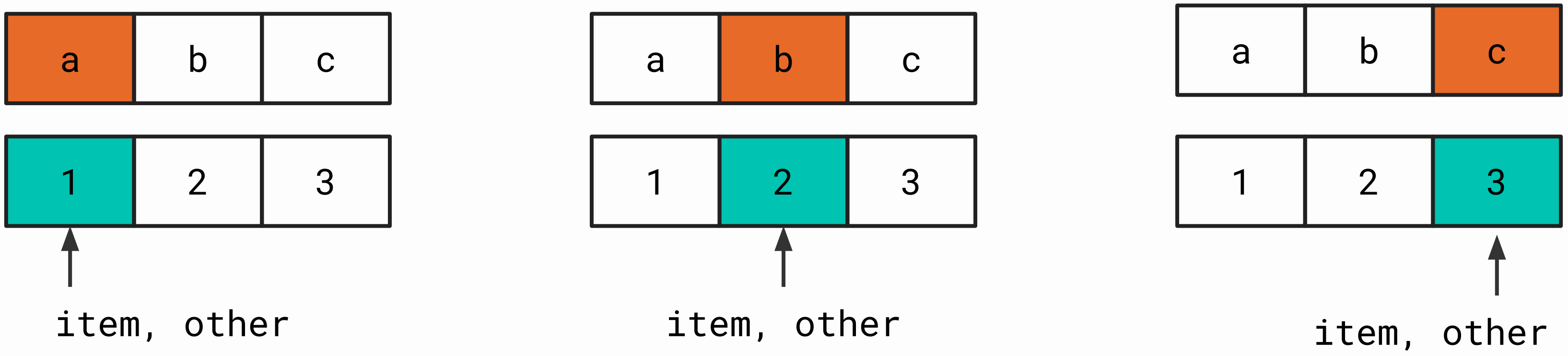
Next, print each color in this format:

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

Multiple Looping

You can access two items at once from two different sequences using the zip function

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



Multiple Loopings Example

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

```python
names = ('Client 1', 'Client 2', 'Client 3')
balances = (10_000, 20_000, 3_000)
ids = (1, 2, 3)
```

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

Quick Exercise: Student Records

Given the two tuples

```
student_names = (“Juan”, “Maria”, “Joseph”)
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.
```

Challenge: Print the highest scorer

Enumerate Looping

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

```
items = (‘a’, ‘b’, ‘c’)
for index, items in enumerate(items):
    print(index, items)
```

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

Enumerate Looping (Different Start)

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

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

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

Quick Exercise: Attendance Log

Given the following tuple

```
student_names = (“Juan”, “Maria”, “Joseph”)
```

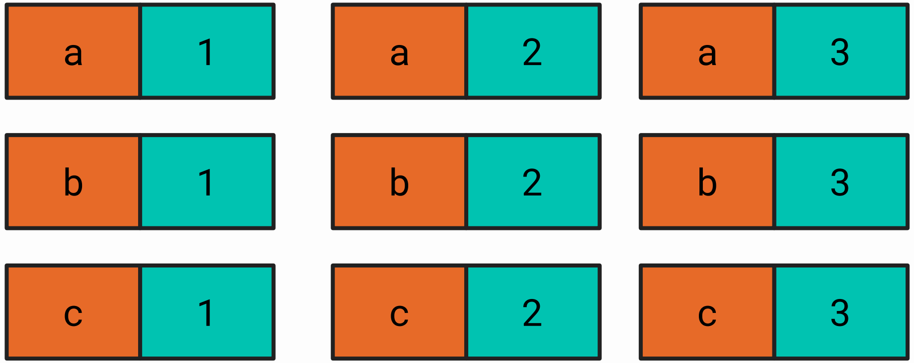
Print the student names in the following format

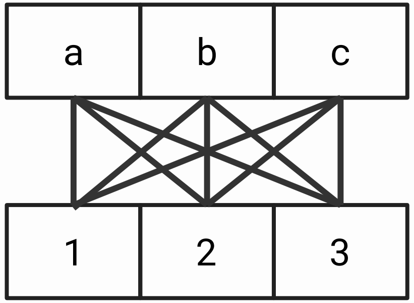
```
Attendance Log:
    Student 1: Juan
    Student 2: Maria
    Student 3: Joseph
```

Nested Looping

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

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





Quick Exercise: Meeting Generation

Given the two tuples, create every possible pairing of **developer** and **tester**

```
developer = ("Developer 1", "Developer 2", "Developer 3")
tester = ("Tester 1", "Tester 2", "Tester 3")
```

```
Developer 1 - Tester 1
Developer 1 - Tester 2
Developer 1 - Tester 3
Developer 2 - Tester 1
Developer 2 - Tester 2
Developer 2 - Tester 3
Developer 3 - Tester 1
Developer 3 - Tester 2
Developer 3 - Tester 3
```

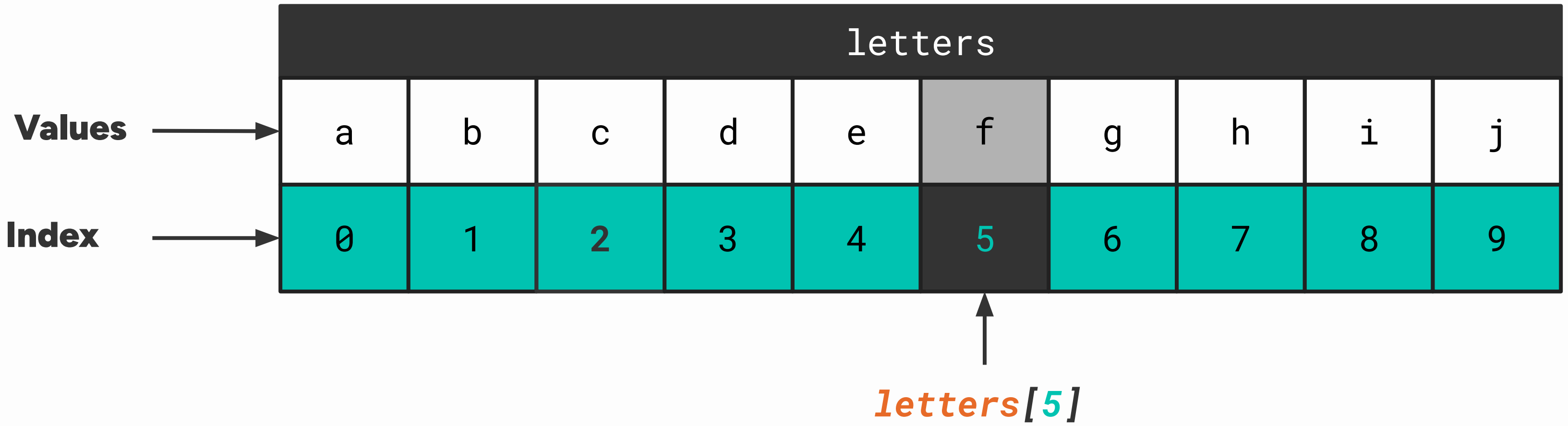
Index Logic

Always remember to start at zero

Item Access

Specific values can be accessed in a sequence using the index operation

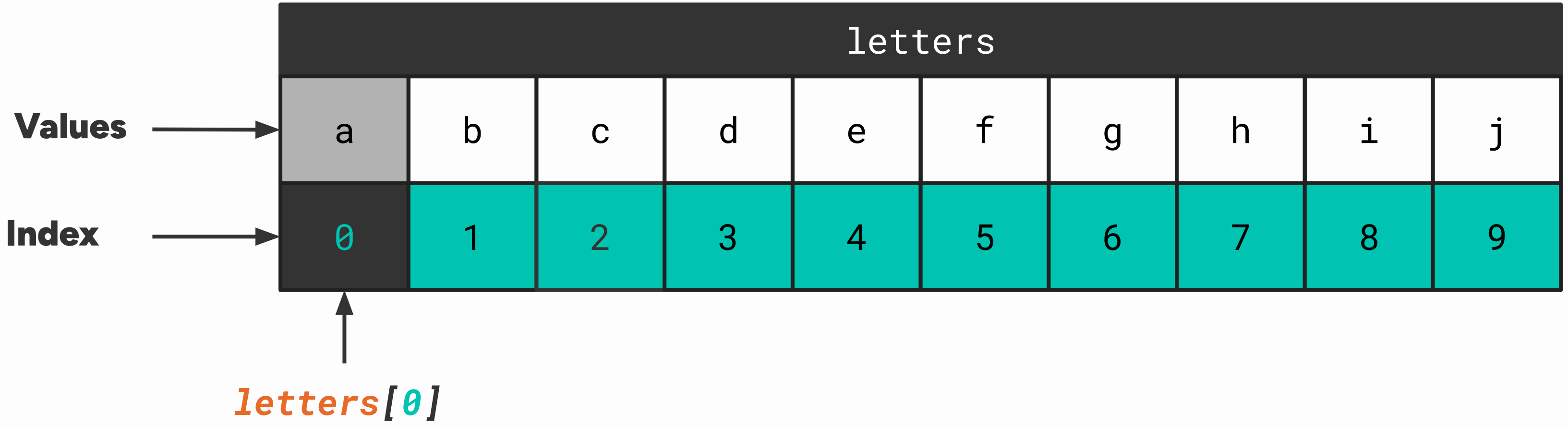
```python
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
print(letters[5])
```



Item Access

Specific values can be accessed in a sequence using the index operation

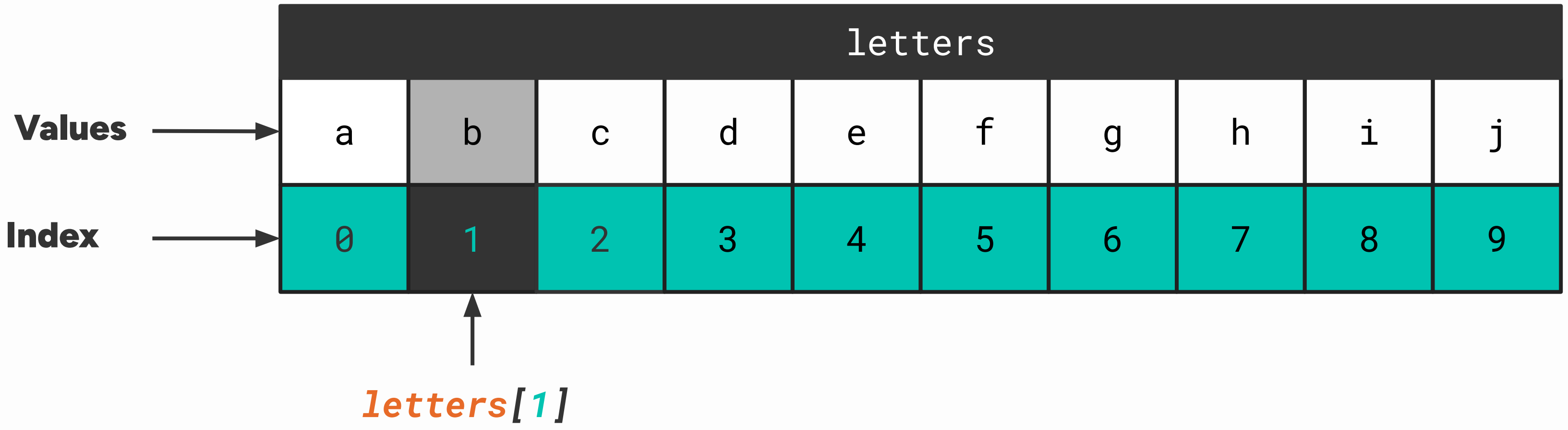
```
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
print(letters[0])
```



Item Access

Specific values can be accessed in a sequence using the index operation

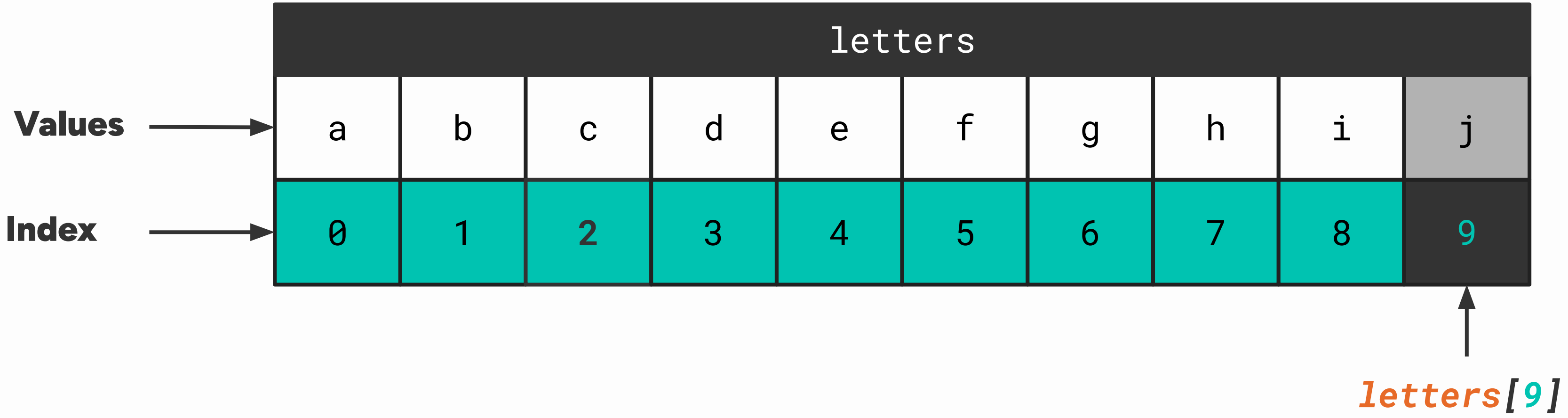
```
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
print(letters[1])
```



Item Access

Specific values can be accessed in a sequence using the index operation

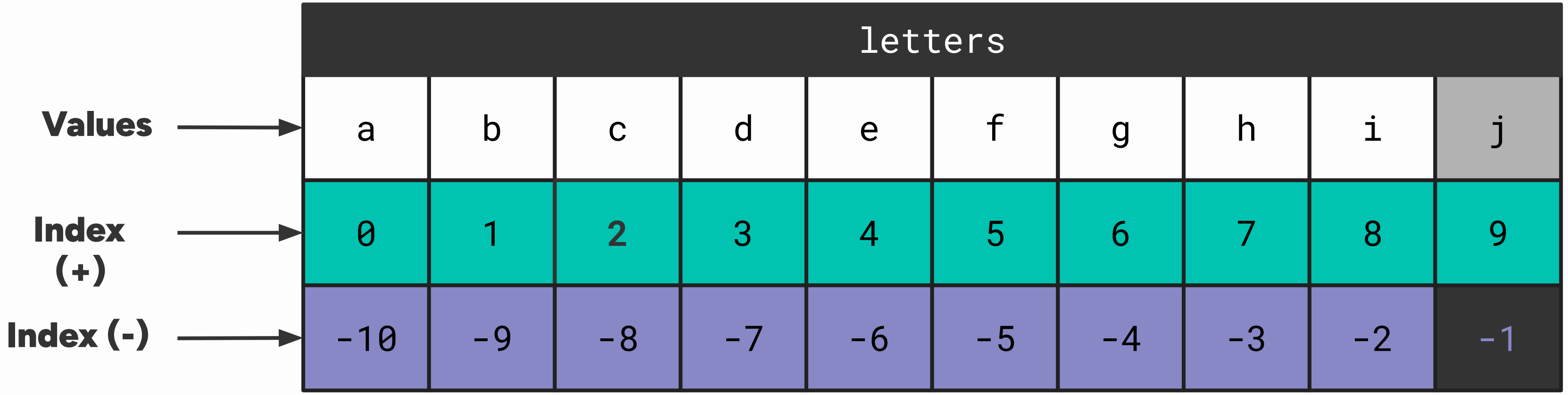
```
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
print(letters[9])
```



Item Access

Specific values can be accessed in a sequence using the index operation

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



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

Item Access

Specific values can be accessed in a sequence using the index operation

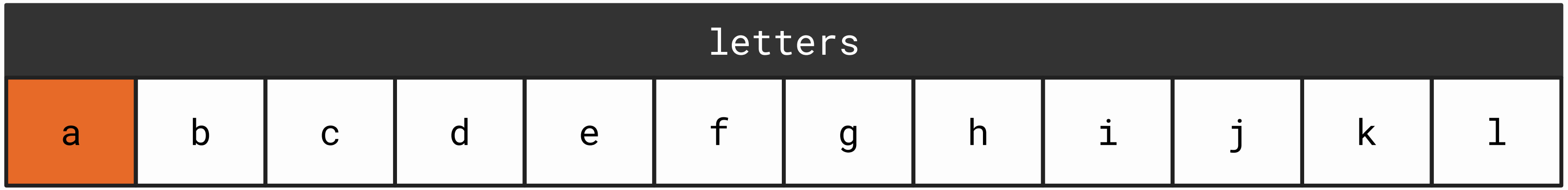
```
letters = (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’)
print(letters[-2])
```

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

ASAN NGA BA AKO SAYO?

Wish fm 107.5

Find the Index

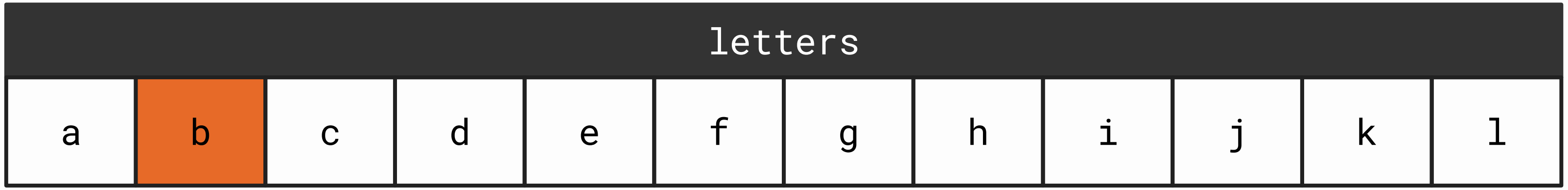


Find the Index

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

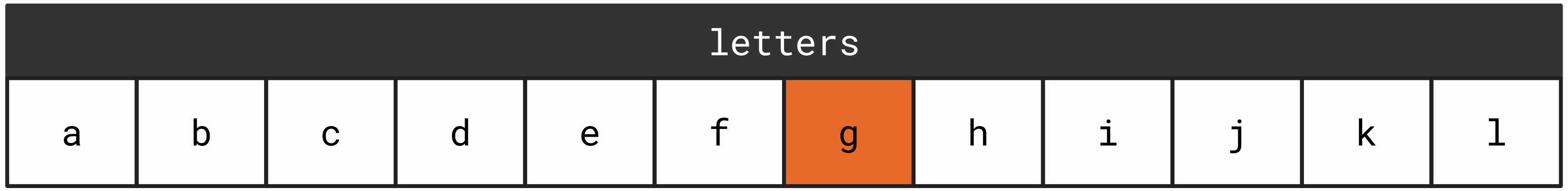
letters											
a	b	c	d	e	f	g	h	i	j	k	l

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 tuple

```
letters = (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’)
```

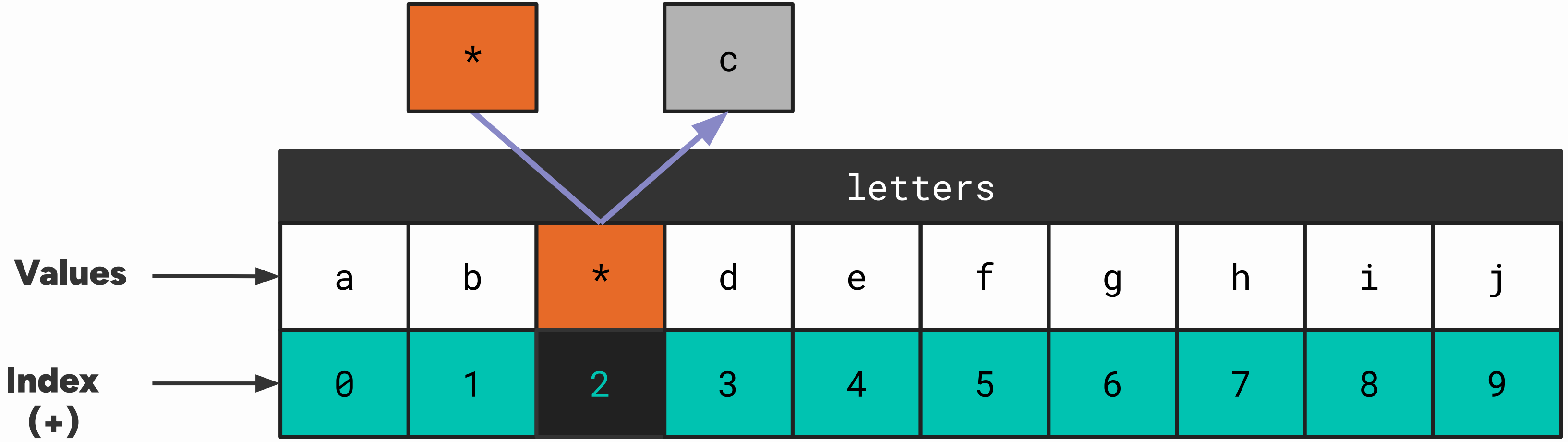
Print the first, third, and fifth letter to create the following output

```
a
c
e
```

Item Modification

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

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

Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

```
student_data = [(“Maria”, 98), (“Pedro”, 30), (“Bax”, 10)]
```

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

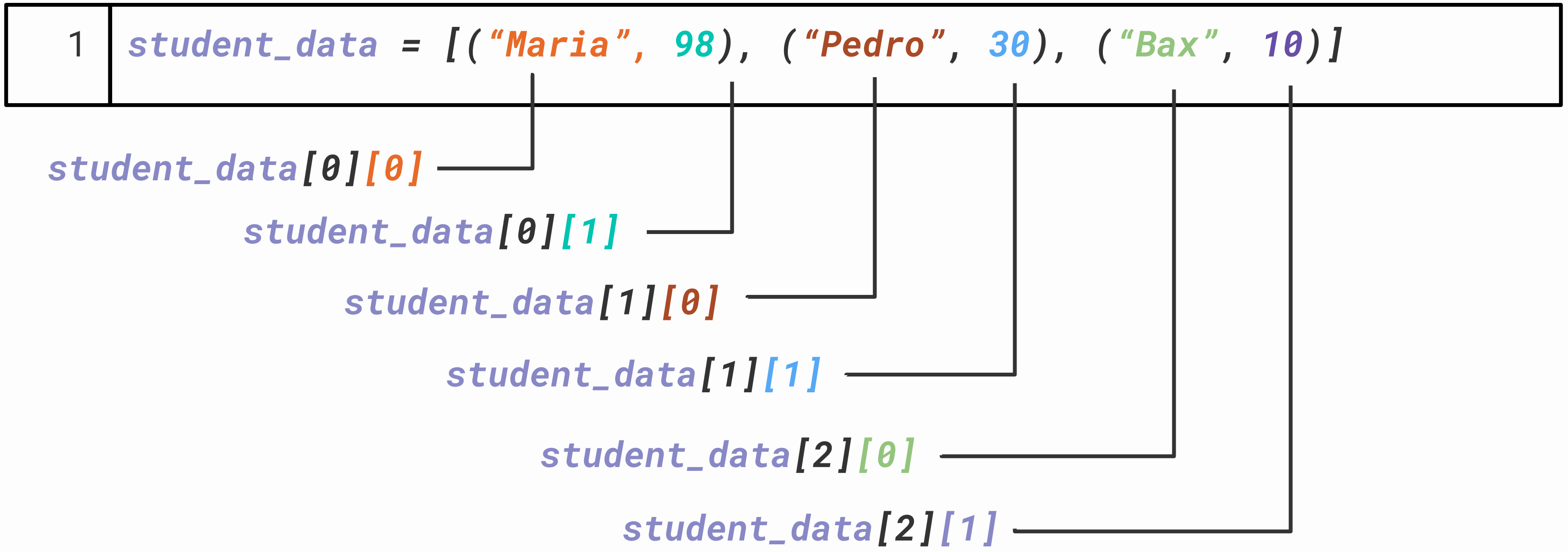
```
first_record = student_data[0]
first_record_score = first_record[1]
```

You can also directly access it by chaining indexing immediately

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

Group Inside a Group

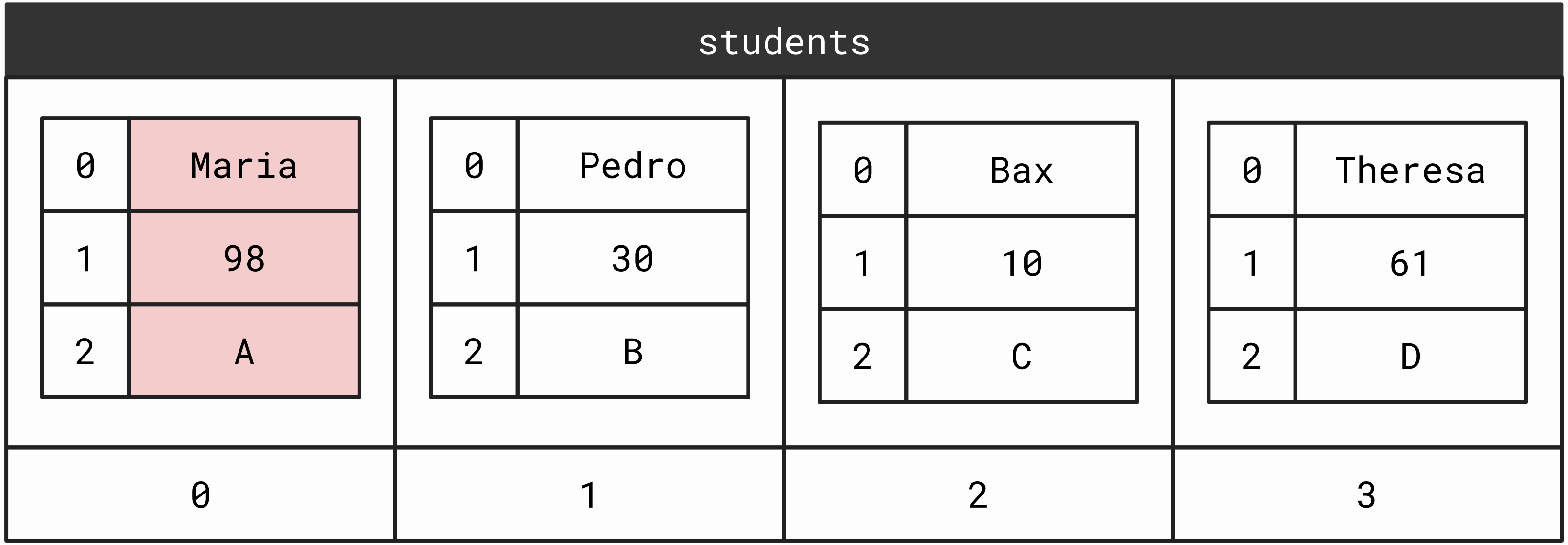
Lists and tuples can also contain lists or tuples inside them



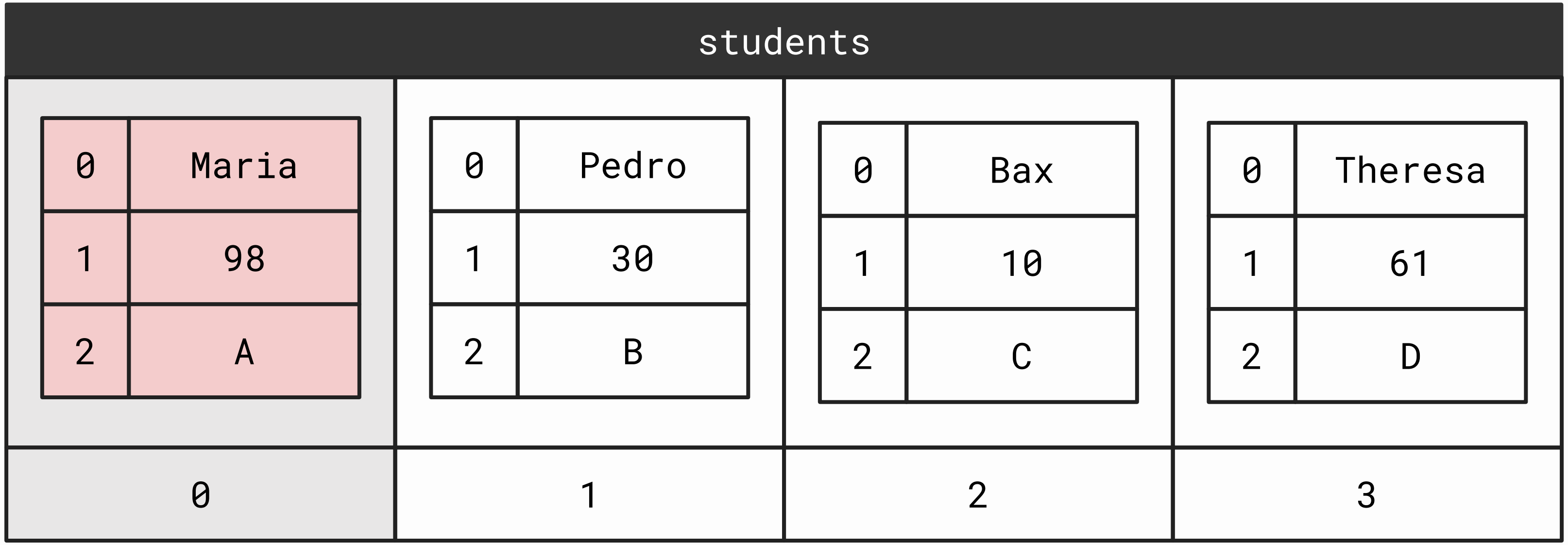
```
student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```

NASAAN KA NA
PolyEast RECORDS
LYRIC VIDEO

Find the Index

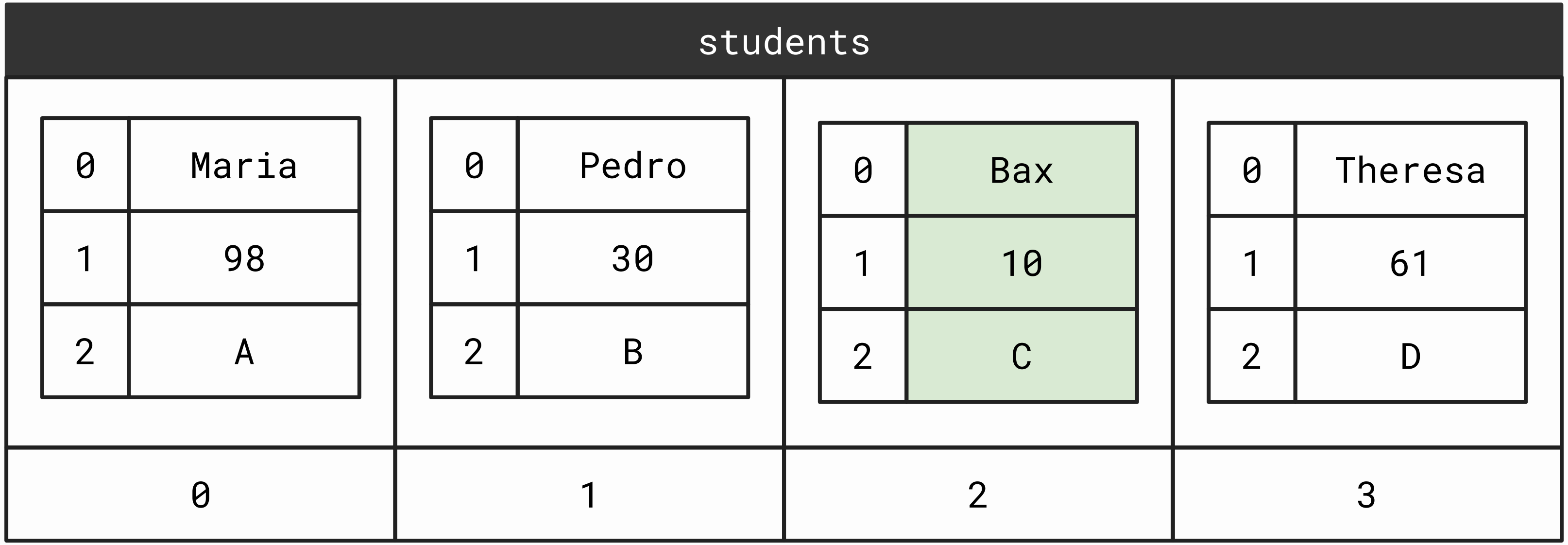


Find the Index

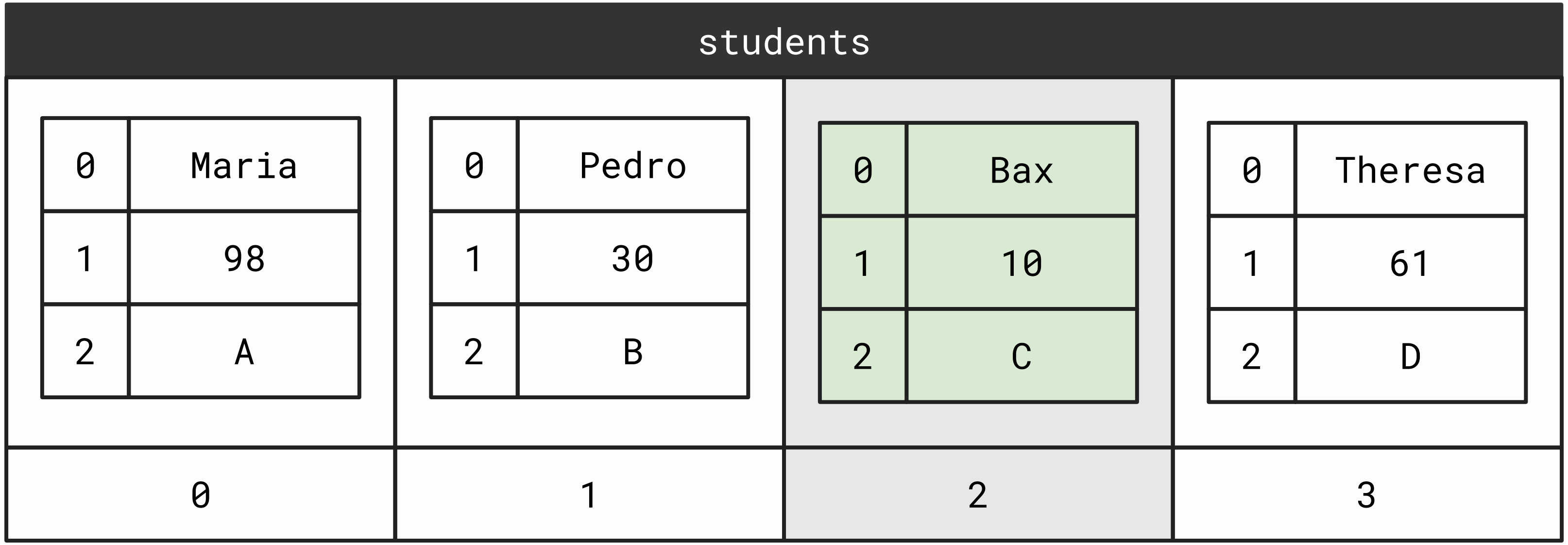


students [0]

Find the Index

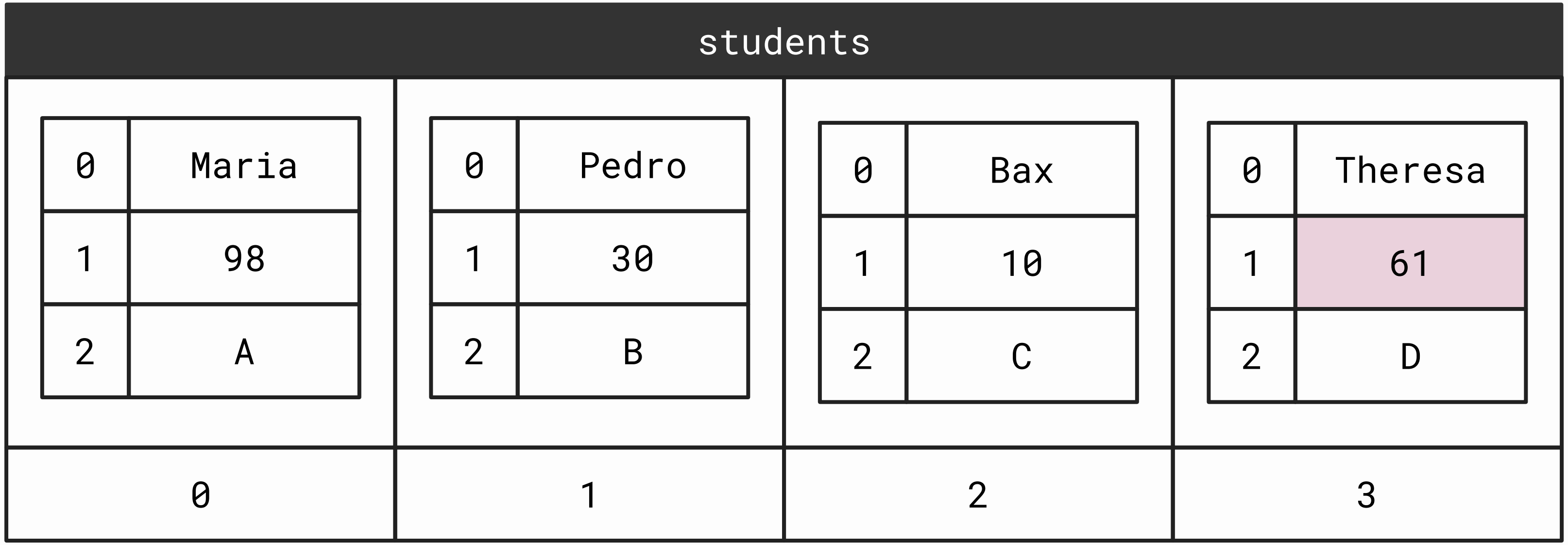


Find the Index

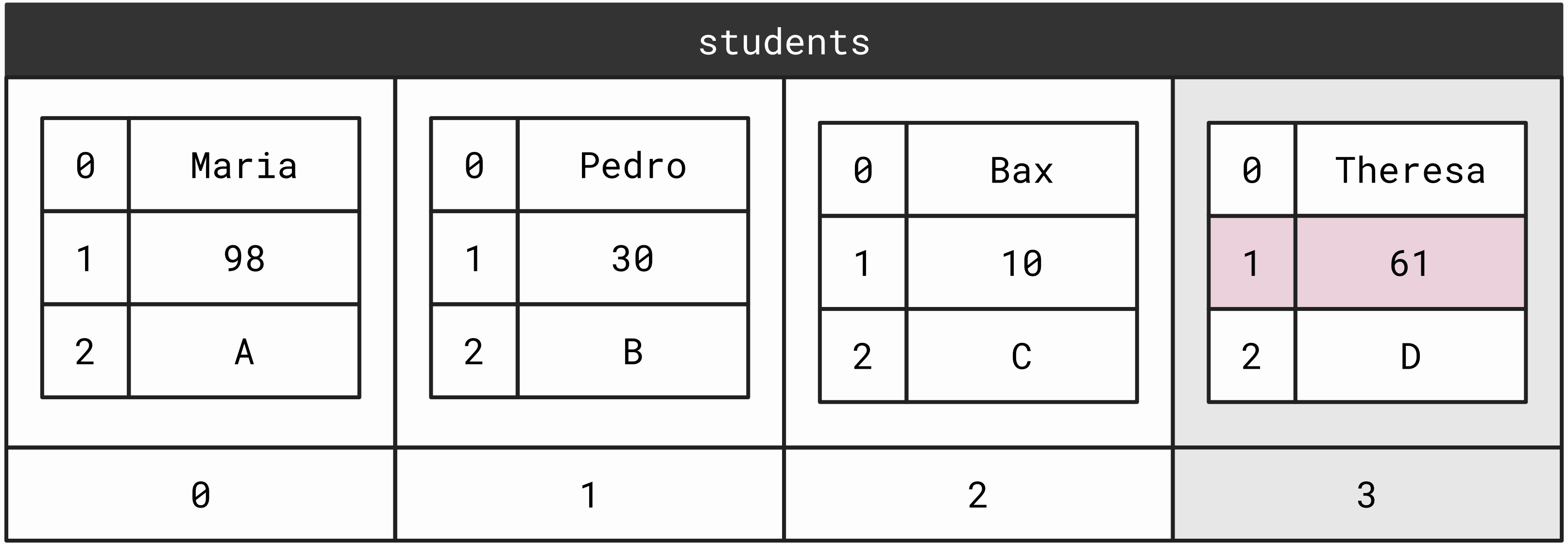


students [2]

Find the Index

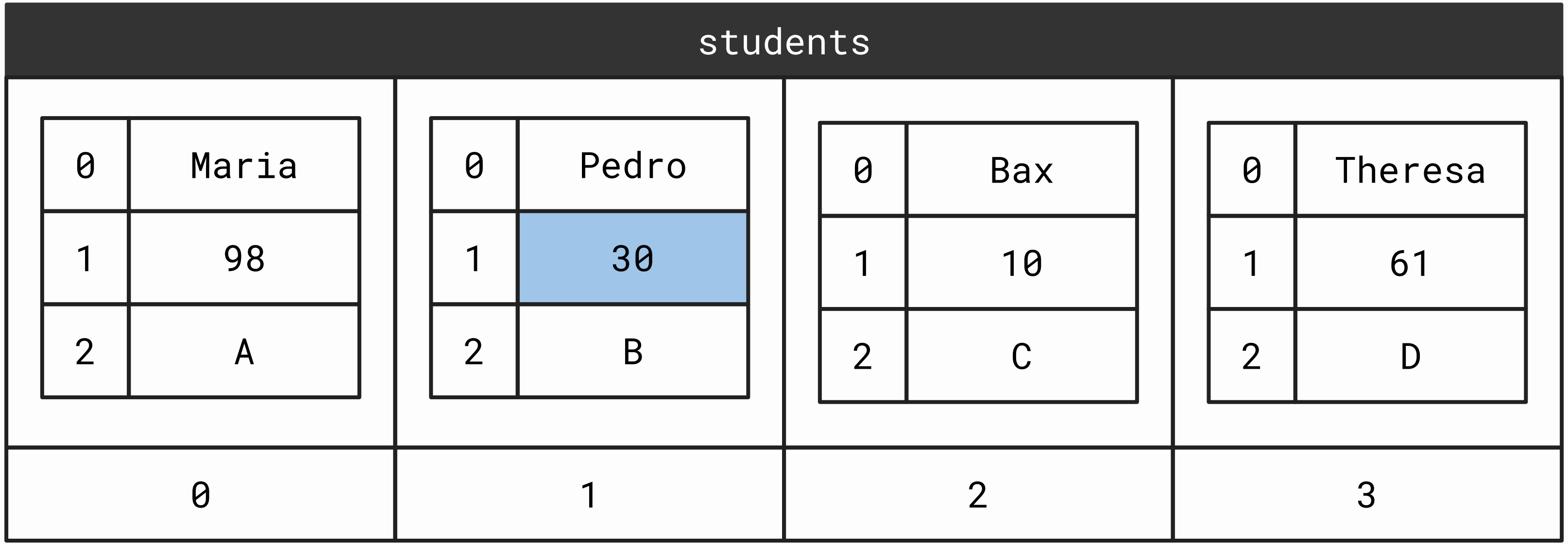


Find the Index

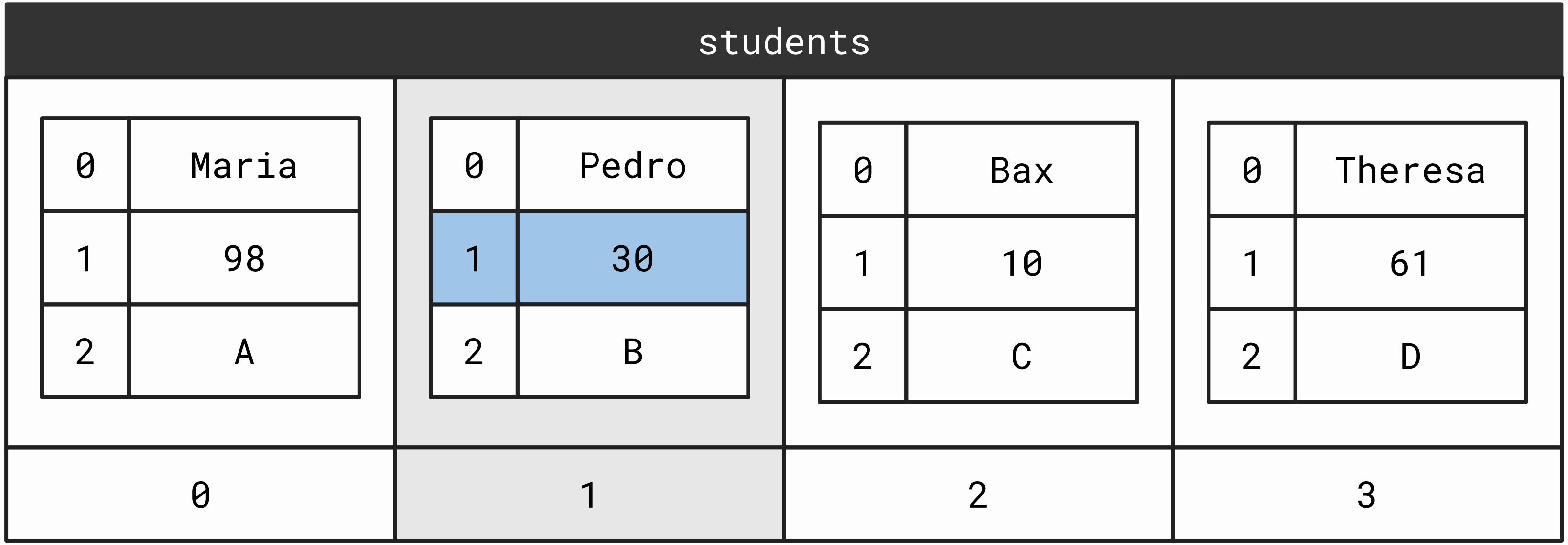


students [3][1]

Find the Index



Find the Index



students [1][1]

Slicing

Indexing for more than one element

Slicing [Start:End]

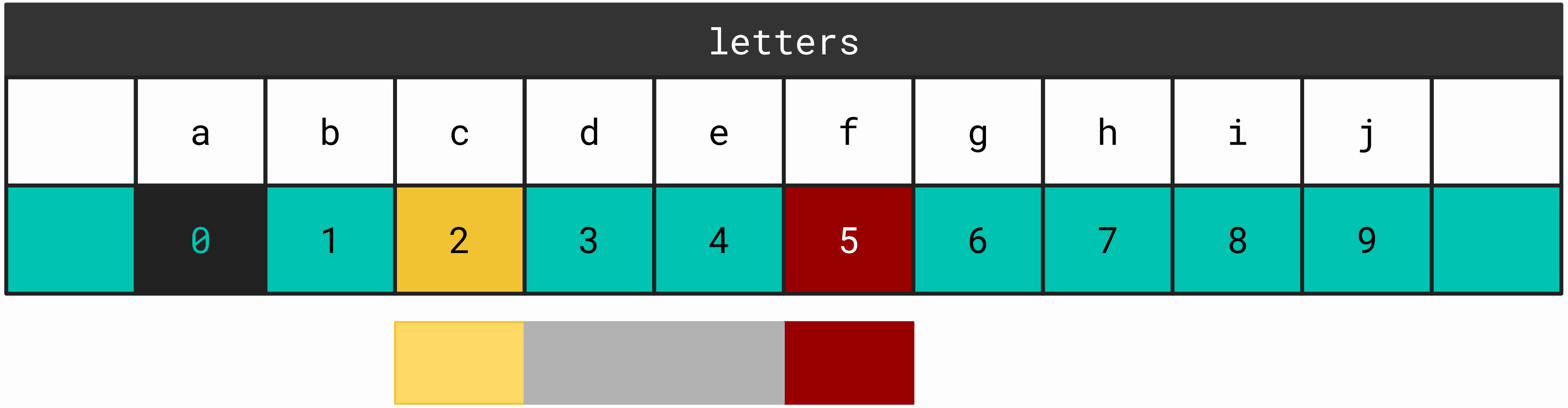
Sequences can index multiple items using the slicing

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

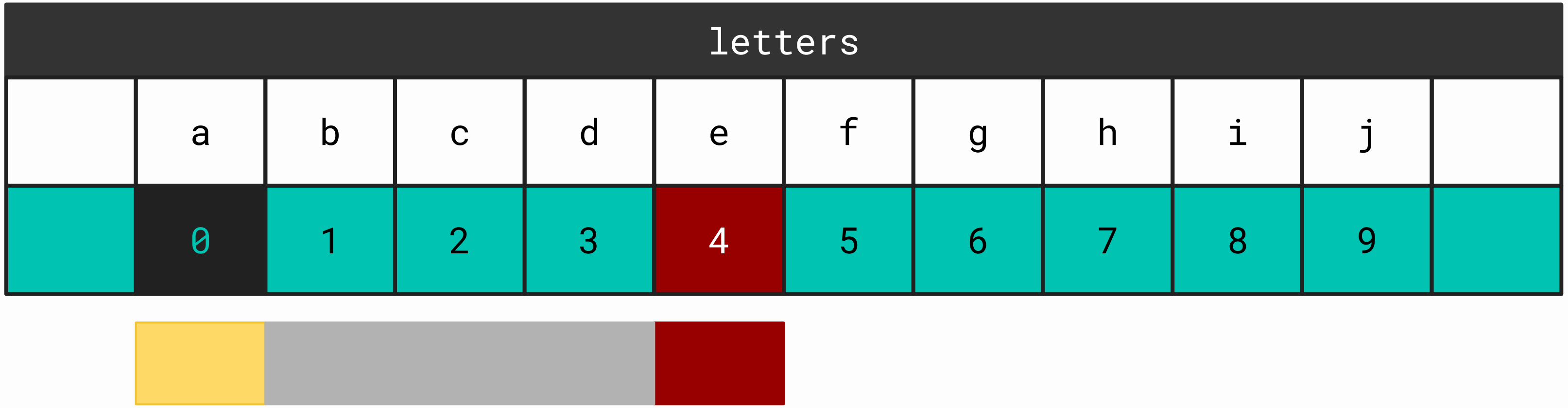
```
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
letters[2:5]
```



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

Example 2

```
letters = (‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’)
letters[:4]
```



[‘a’, ‘b’, ‘c’, ‘d’]

Example 3

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

Quick Exercise: Letter Slicing

Given the following tuple

```
letters = (‘f’, ‘o’, ‘r’, ‘e’, ‘g’, ‘o’, ‘n’, ‘e’)
```

Print the following sublists

```
[‘f’, ‘o’, ‘r’, ‘e’]
[‘e’, ‘g’, ‘o’]
[‘g’, ‘o’, ‘n’, ‘e’]
```

Slicing [Start:End:Step]

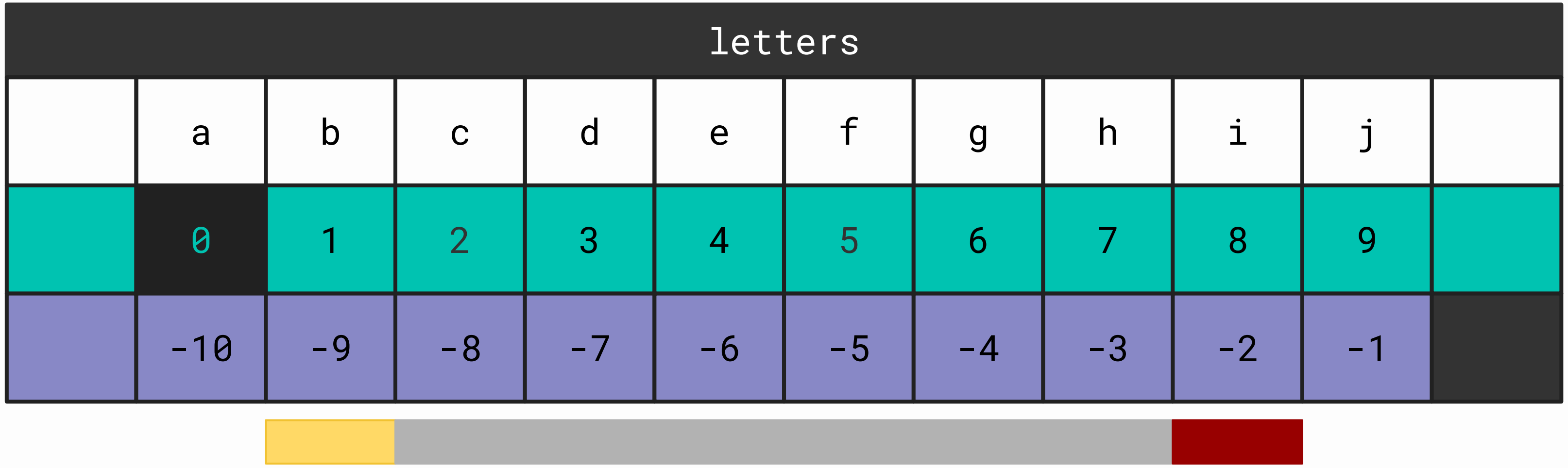
Sequences can index multiple items using slicing

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

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

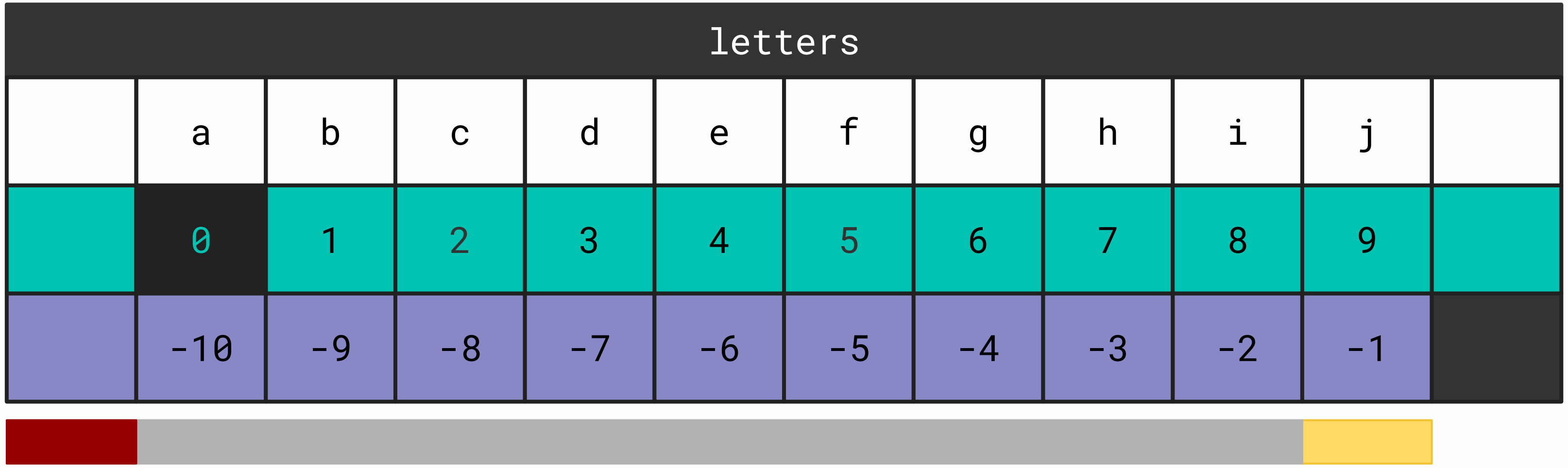


letters											
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

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

Example 2

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

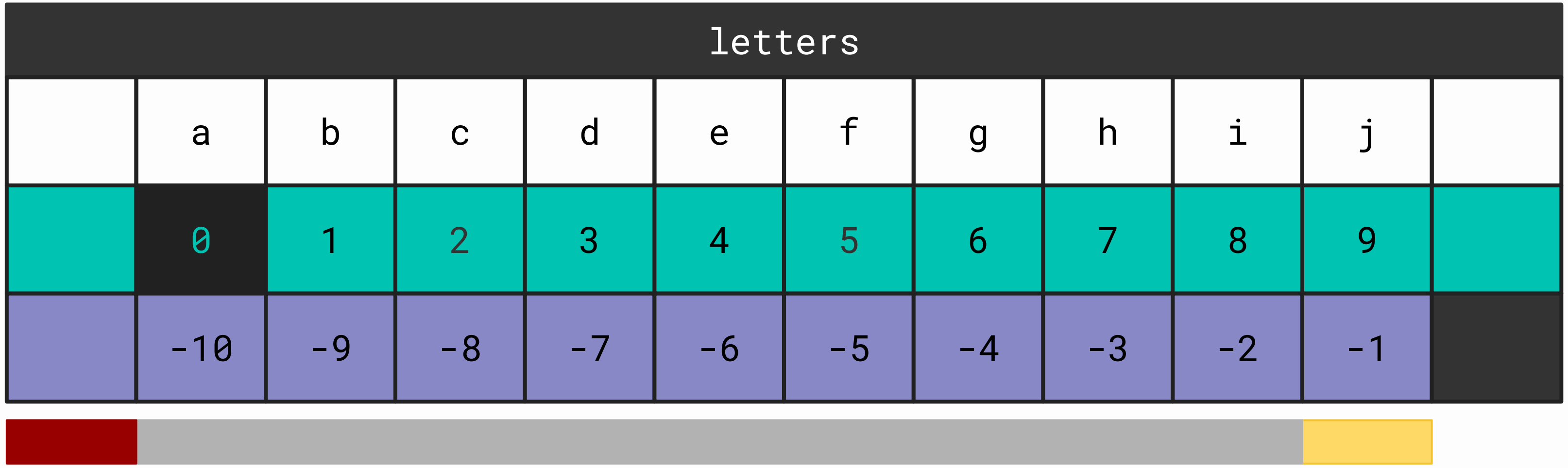


letters											
	a	b	c	d	e	f	g	h	i	j	
	0	1	2	3	4	5	6	7	8	9	
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

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

Example 3

```python
letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')
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 tuple

```
letters = (‘f’, ‘r’, ‘i’, ‘e’, ‘n’, ‘d’)
```

Print the following sublists

```
[‘f’, ‘i’, ‘n’]
[‘r’, ‘e’, ‘d’]
```

Sequence Functions

Convenient functions for list and tuples

Min Function

The **`min`** function returns the smallest value in a given sequence

```python
example = (1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

```python
print(min(example))
print(example)
```

```
1
(1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

Max Function

The **max** function returns the largest value in a given sequence

```
example = (1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

```
print(max(example))
print(example)
```

```
7
(1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

Sum Function

The **sum** function returns the total if all the items in a given sequence were added together

```
example = (1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

```
print(sum(example))
print(example)
```

```
30
(1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

Length Function

The **`len`** function returns the number of items in a given sequence

```
example = (1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

```
print(len(example))
print(example)
```

```
10
(1, 3, 3, 5, 6, 7, 1, 2, 1, 1)
```

Quick Exercise: Class Statistics

Given the following scores

```python
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
```

Sorted Function (Ascending)

The **`sorted`** function returns a copy of the input but sorted in ascending order

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

```
print(sorted(example))
print(example)
```

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

Sorted Function (Descending)

The **`sorted(reverse=True)`** function returns a copy of the input but sorted (descending)

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

```python
print(sorted(example, reverse=True))
print(example)
```

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

Reversed Function

The **reversed** function returns a copy of the input but with the items in reversed order

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

```
print(reversed(example))
print(example)
```

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

Quick Exercise: Student Rankings

Given the following scores

```
student_scores = (98, 75, 100, 86, 100, 3)
```

Print the scores in `student_scores`, but in order of greatest to least.

```
Rank 1: 100
Rank 2: 100
Rank 3: 98
Rank 4: 86
Rank 5: 75
Rank 6: 3
```

List Methods

Functions for dynamic sequences

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)
```

Recall: Functions can’t write outside

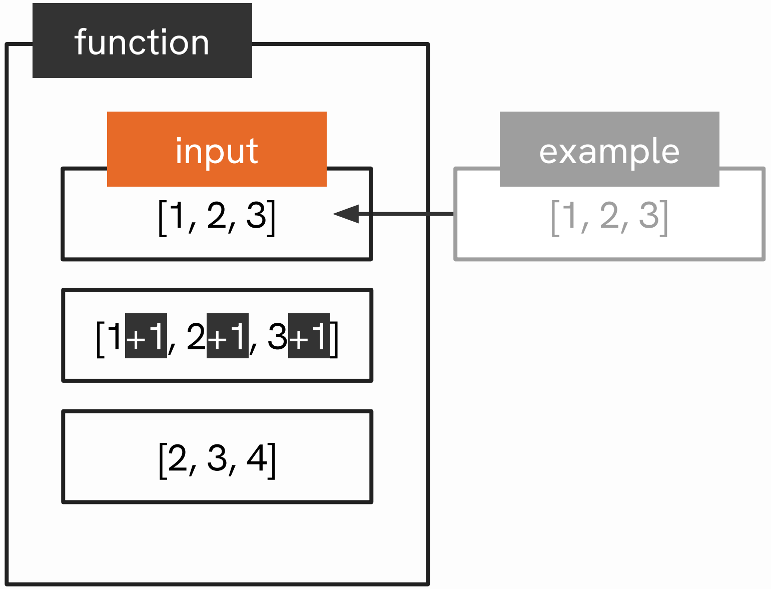
This mainly because the syntax for updating variables is the same syntax for making a new one. In here, we’re making a new list with the same variable name

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

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

Functions

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

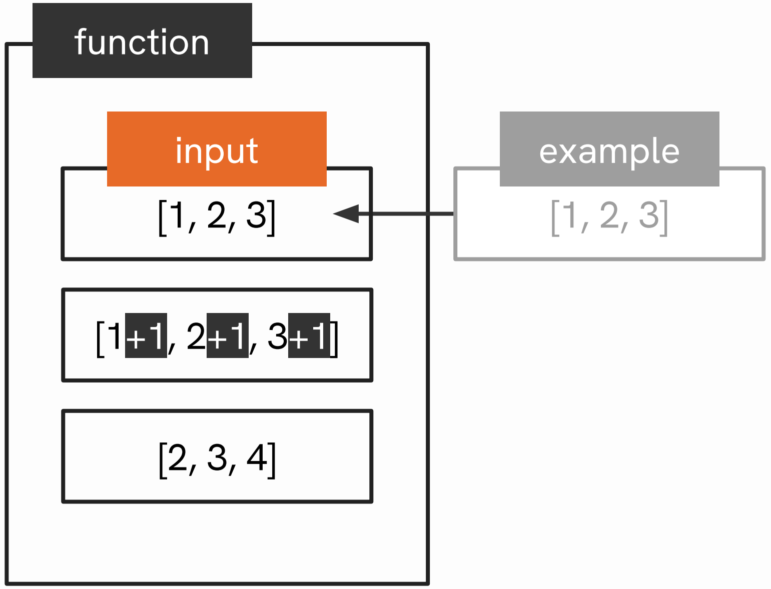


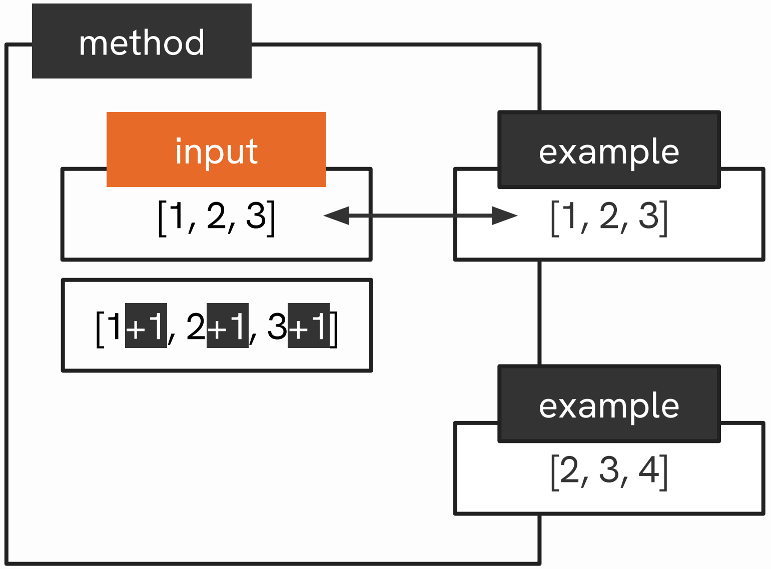
However, methods can!*

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)
```

```
[1, 2, 3]
[1, 2, 3, 999]
```

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)
```

```
[1, 2, 3]
[1, 2, 3, 999]
```

Append Method

The **append** method adds an item to the end of a given list

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

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

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

Extend Method

The **extend** method adds all the items of the input to the end of a list

```
example = [1, 3, 3, 5, 4]
extensions = (999, -1, 0)
```

```
example.extend(extensions)
print(example)
```

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

Insert Method

Given an index, the **`insert`** method adds an item to that position in the list

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

```python
example.insert(0, 999)
print(example)
```

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

Quick Exercise: Running List

Given the empty list

```
attendee_names = []
```

Ask the user for an integer

```
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`

Remove Method

The **`remove`** method removes an item from the given list. This raises an error if item not found.

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

```python
example.remove(5)
print(example)
```

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

Safe Remove Method

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

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

```
item_to_remove = 999
if item_to_remove in example:
    example.remove(item_to_remove)
print(example)
```

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

Clear Method

The **`clear`** method removes all items in a list

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

```python
example.clear()
print(example)
```

```
[]
```

Pop Method

Given an index, the **pop** method removes the item in that position from the given list list.
If the index is invalid, this will raise an error.

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

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

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

Pop Method with Return

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

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

```
removed_item = example.pop(-1)
print(removed_item)
print(example)
```

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

Quick Exercise: Late Attendee

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

```
Late Attendee: Attendee Name
```

Sets

Collection of unordered items

Set Definition

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

```
letters = {‘a’, ‘a’, ‘b’, ‘c’, ‘d’}
print(letters)
```

letters
d, c, a, b

Mutable Instances

Sets can only use static or non-mutable data types

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

Set Add Method

The **add** method adds an item to the given set

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

```
print(example)
example.add(99)
print(example)
```

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

Set Add Method - Uniqueness

If the item already exists in the set, the item is ignored.

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

```
print(example)
example.add(3)
print(example)
```

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

Quick Exercise: Unique Running List

Given the empty set

```
attendee_names = set()
```

Ask the user for an integer

```
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

The **discard** method removes an item from the given set

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

```python
print(example)
example.discard(5)
print(example)
```

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

Set Discard Method - Missing

If the item does not exist in the set, nothing happens

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

```python
print(example)
example.discard(99)
print(example)
```

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

Quick Exercise: Remove Special Case

From the ***unique running list exercise***, remove the attendees with the same name as you

```python
attendee_names = set()
attendee_count = int(input(“How many attendees? “))
```

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

```
Removed: Attendee Name
```

Set Pop Method

The **pop** method randomly removes an item from the set. The removed item can be retrieved.

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

```python
print(example)
return_value = example.pop()
print(example)
print(return_value)
```

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

Quick Exercise: Raffle Winner

From the ***Remove Special Case exercise***, pick a random attendee as a raffle winner!

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

```
attendee name:
attendee 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)

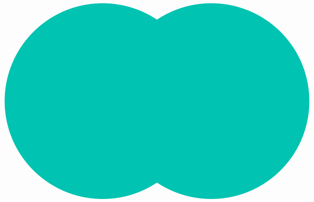
Set Operations

Based on the mathematical set operations

Set Union

```
set1 = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’}
set2 = {‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’}
print(set1.union(set2))
print(set1 | set2)
```

Set1					
a	b	c	d	e	f

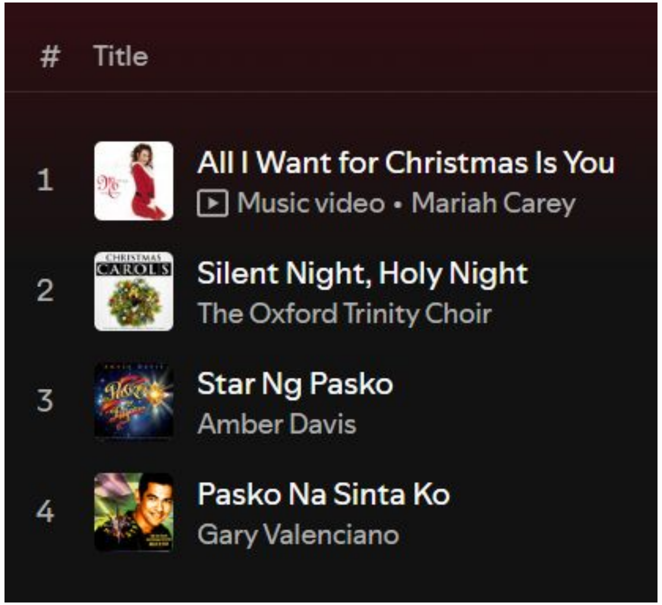


Set 2					
d	e	f	g	h	i

Quick Exercise: Combined Playlist

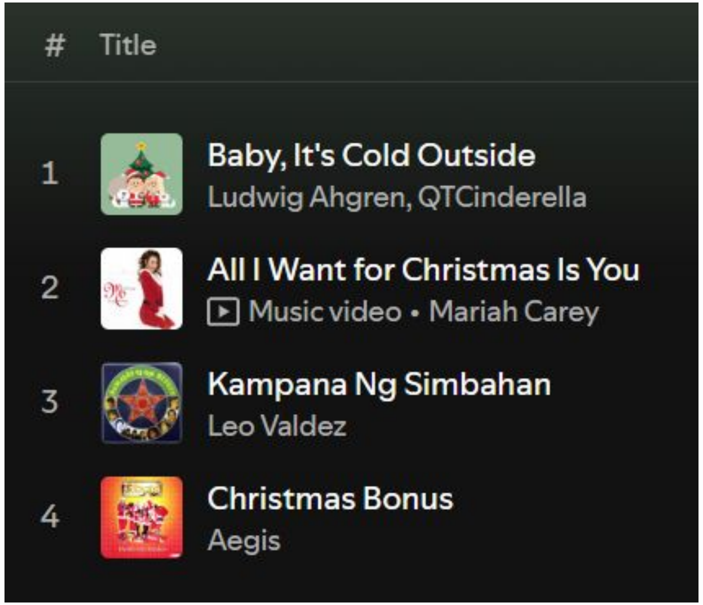
Create a new playlist that combines all the songs

Your Playlist



#	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



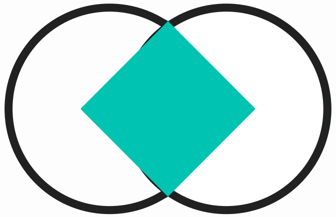
#	Title
1	Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	All I Want for Christmas Is You Music video • Mariah Carey
3	Kampana Ng Simbahan Leo Valdez
4	Christmas Bonus Aegis

Set Intersection

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

Set1					
a	b	c	d	e	f

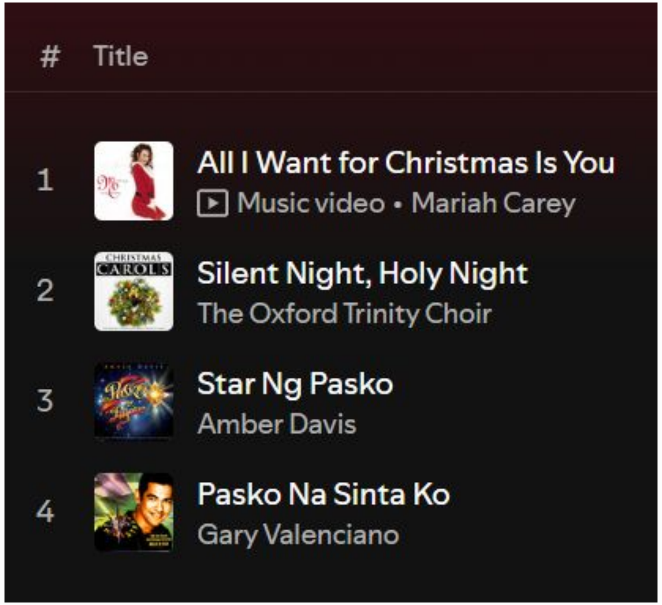
Set 2					
d	e	f	g	h	i



Quick Exercise: Mutual Playlist

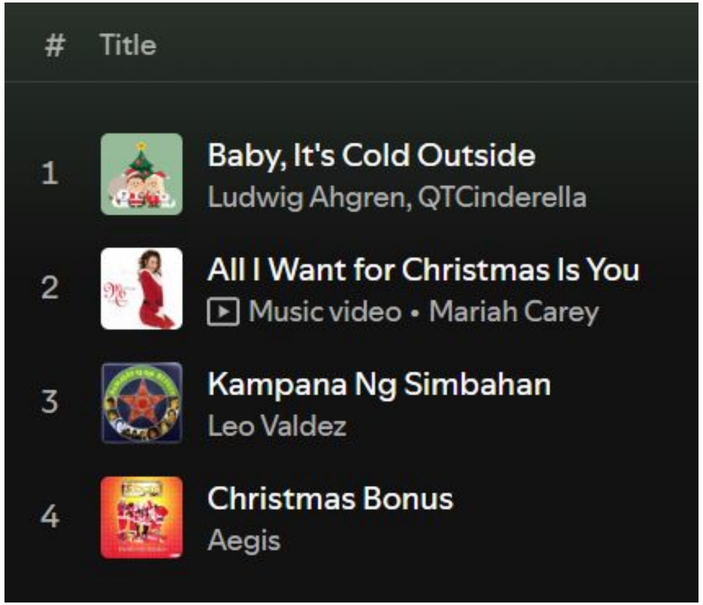
Create a new playlist for songs that are in both playlist

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

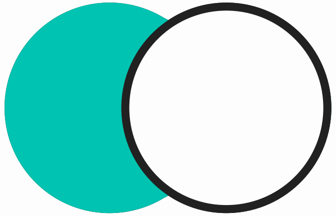


#	Title
1	Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	All I Want for Christmas Is You Music video • Mariah Carey
3	Kampana Ng Simbahan Leo Valdez
4	Christmas Bonus Aegis

Set Difference

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

Set1					
a 🟩	b 🟩	c 🟩	d	e	f

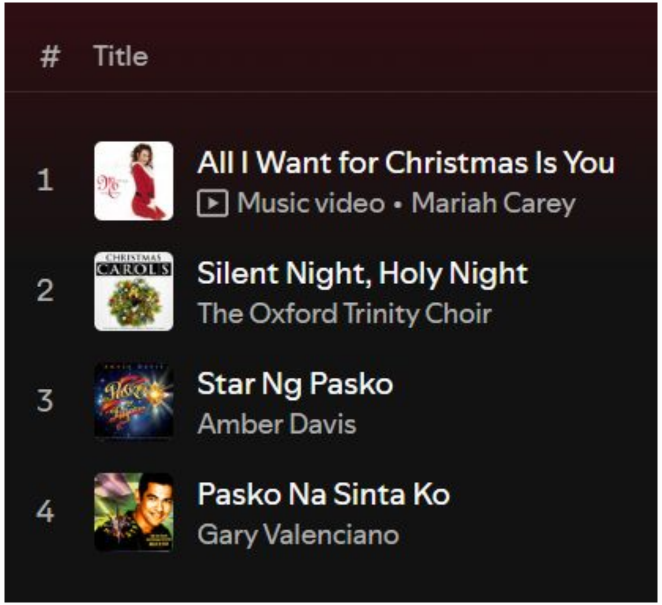


Set 2					
d	e	f	g	h	i

Quick Exercise: Your Unique Songs

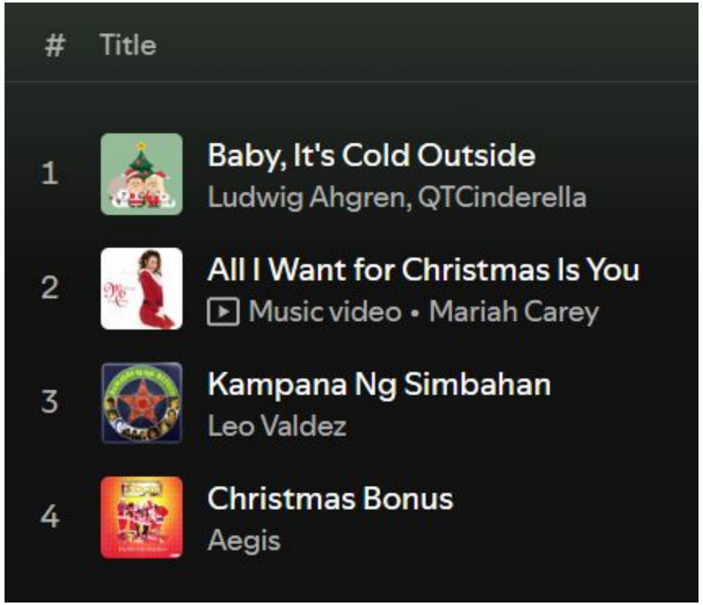
Create a new playlist for songs that only you have

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

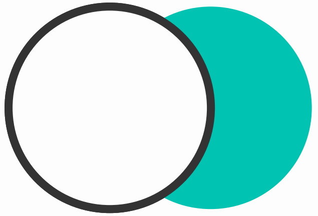


#	Title
1	Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	All I Want for Christmas Is You Music video • Mariah Carey
3	Kampana Ng Simbahan Leo Valdez
4	Christmas Bonus Aegis

Set Difference (Order Matters)

```
set1 = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’}
set2 = {‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’}
print(set2.difference(set1))
print(set2 - set1)
```

Set1					
a	b	c	d	e	f

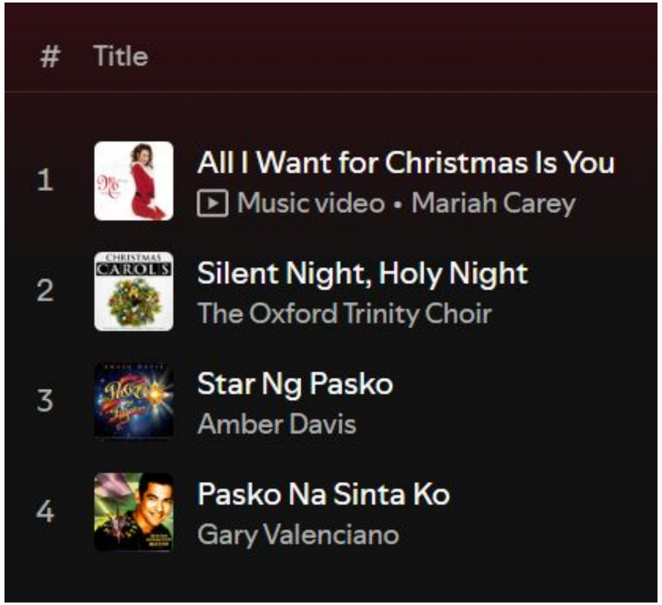


Set 2					
d	e	f	g	h	i

Quick Exercise: Friend Unique Songs

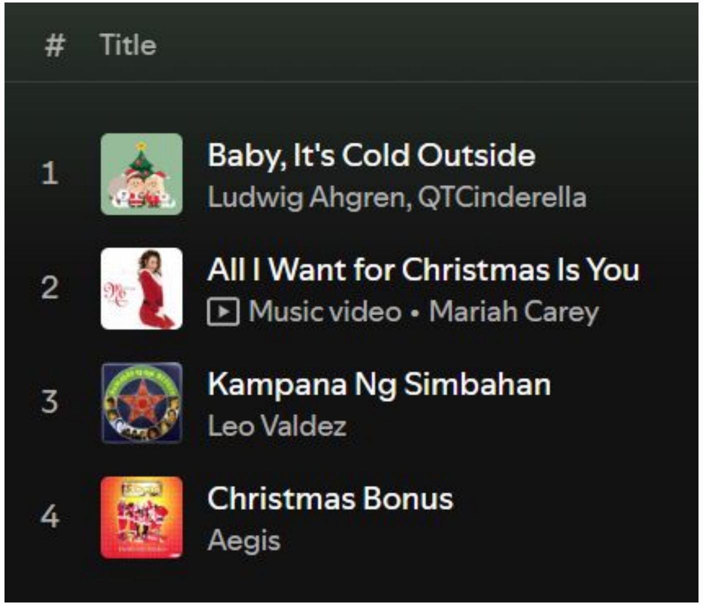
From the previous exercise, create a new playlist for songs that only your friend has

Your Playlist



#	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

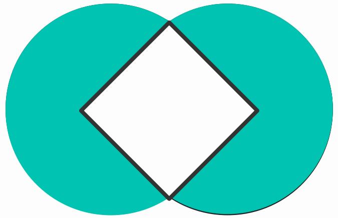


#	Title
1	Baby, It's Cold Outside Ludwig Ahgren, QTCinderella
2	All I Want for Christmas Is You Music video • Mariah Carey
3	Kampana Ng Simbahan Leo Valdez
4	Christmas Bonus Aegis

Set Symmetric Difference

```python
set1 = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, ‘g’}
set2 = {‘d’, ‘e’, ‘f’, ‘g’, ‘h’, ‘i’, ‘j’}
print(set1.symmetric_difference(set2))
print(set1 ^ set2)
```

Set1					
a	b	c	d	e	f

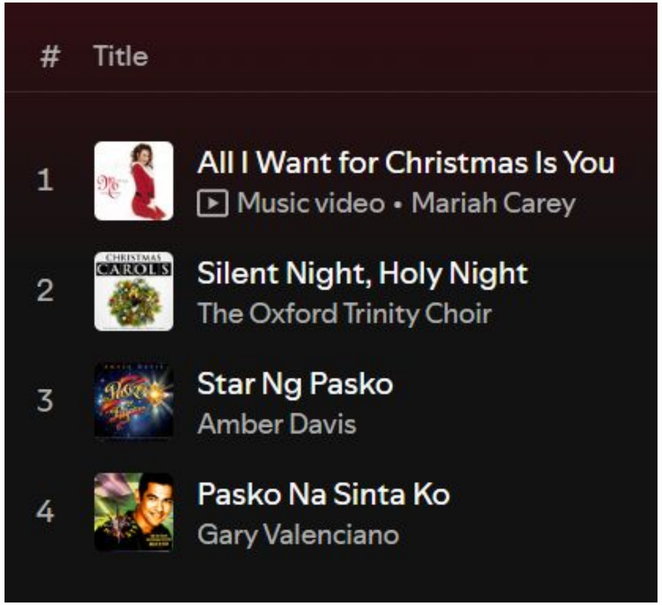


Set 2					
d	e	f	g	h	i

Quick Exercise: Unique Songs

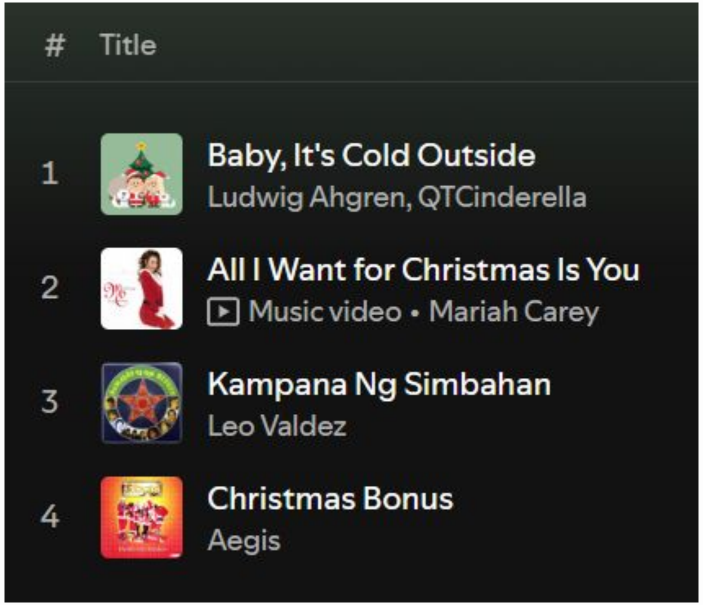
Create a new playlist for songs that is not mutual

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 mapping data

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

$$\begin{tikzcd}\boxed{\texttt{key}} \arrow[r] & \boxed{\texttt{value}}\end{tikzcd}$$

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}
```

Quick Exercise: Favorites

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

```
favorites = {
    “number”: 5,
    “color”: “black”,
}
```

Next, print the `favorites` dictionary

```
print(favorites)
```

Dictionary Access

The dictionary values can be accessed using the same syntax as a list

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

print(student_records[“Joseph”])
```

```
81
```

Dictionary Access (Safe)

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

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

print(student_records.get(“Elizabeth”))
```

```
None
```

Mapping

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

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

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

```

Quick Exercise: Extend Codes

Add more country codes

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

code = input(“Enter country code: “)
print(f"{code} -> {country_codes.get(code)}")
```

Dictionary Parts

Iteration options for dictionaries

Dictionary Iteration (Keys)

The **keys** method returns a sequence of the dictionary keys

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

for student_name in student_records.keys():
    print(student_name)
```

Dictionary Iteration (Implicit Keys)

However, by default, using iteration and the **in** operator with the dict itself directs to its keys

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

for student_name in student_records:
    print(student_name)
```

Quick Exercise: Key Iteration

Using your `favorites` dict in the ***Favorites*** exercise, print each key in this format:

```
My Favorites:
    Number
    Color
```

Dictionary Iteration (Values)

The **`values`** method returns a sequence of the dictionary values

```python
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

for student_score in student_records.values():
    print(student_score)
```

Quick Exercise: Value Iteration

Using your `favorites` dict in the ***Favorites*** exercise, print each value in this format:

```
My Favorites:
    2
    Black
```

Dictionary Iteration (Key-Value)

The **`items`** method returns a sequence of tuples, each having two items - a key and its value

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}

for student_name, student_score in student_records.items():
    print(student_name, student_score)
```

Quick Exercise: Complete Iteration

Using your `favorites` dict in the ***Favorites*** exercise, 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

Dictionaries use the index operation to add new entries.

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}
student_records[“Chocolate”] = 25
print(student_records[“Chocolate”])
```

```
25
```

Quick Exercise: Task Tracking

Given the empty dictionary

```
attendees = dict()
```

Ask the user for an integer

```
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 ***attendees***

Dictionary Overwriting

Dictionaries also use the index operation for overwriting values

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}
student_records[“Joseph”] = 100
print(student_records[“Joseph”])
```

```
100
```

Dictionary Overwriting Guard

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

```
student_records = {
     “Juan”: 70,
     “Maria”: 98,
     “Joseph”: 81,
     “Elise”: 80
}
if “Joseph” in student_records:
     print(“Joseph is already recorded!”)
else:
     student_records[“Joseph”] = 100
print(student_records[“Joseph”])
```

```
81
```

Quick Exercise: Salary Update

Given the following dictionary

```python
employee = {'Phoenix': 20_000, 'Alex': 30_000, 'Sydney':40_000}
```

Increase the salary of each employee by ten percent

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`)

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

```python
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

```
for entry in product_catalog:
    name = entry['name']
    description = entry['description']
    price = entry['price']
    stock = entry['stock']

    print(f"{name} ({price}) [{stock}] - {description}")
```

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

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

user_profile [“preferences”]

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']

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]

Quick Exercise: Movie Entry

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

```python
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'],
	},
}
```

List Comprehension

The most efficient way to generate a list of items

List Comprehension

List comprehensions are shortcuts to generate multiple items

```python
example_list = []
for number in range(11):
    example_list.append(number)


```

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

Quick Exercise: Square Generator

Using list comprehensions, update the *squares* function to create *number_count* numbers

```python
def squares(n):
    generated_squares = []
    return generated_squares

number_count = int(input("How many to generate? "))
print(squares(number_count))

```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, ...]
```

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]
```

Quick Exercise: Odd Generator

Using list comprehensions, update the *odds* function to create *number_count* numbers

```
def odds(n):
     generated_odds = []
     return generated_odds

number_count = int(input(“How many to generate? “))
print(odds(number_count))

```

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, ...]
```

Singular Number

The most common use of list comprehensions is to quickly create data in specific formats

```
coordinates_sums = [
    x + y + z
        for x in range(10)
        for y in range(10)
        for z in range(10)
]
print(coordinates_sums)
```

Nested Data

The most common use of list comprehensions is to quickly 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)
```

String Generation

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

```python
coordinates_strings = [
    f"({x}, {y}, {z})"
        for x in range(10)
        for y in range(10)
        for z in range(10)
]
print(coordinates_strings)
```

Data Filtering

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

```
requests = {“Andrew”: 10, “Peddy”: 21, “Alex”: 30}
banned = {“Alex”}

adults = [name for name, age in requests.items() if age >= 18]
print(adults)

allowed = [name for name in adults if name not in banned]
print(allowed)
```

Clean Comprehension Format

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

```
def process(number):
    return ((1 + number) // 2)** 3

def condition(number):
    return number > 10

numbers = [991, 12, 89, 34, 121, 0]
data = [process(number) for number in numbers if condition(number)]
print(data)
```

Comprehension +

Additional options for comprehensions

Set Comprehensions

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

```
example_set = set()
text = “I am an igloo!”
for char in text:
    if char not in “.!@#$%^&*()”:
        example_set.append(char)
```

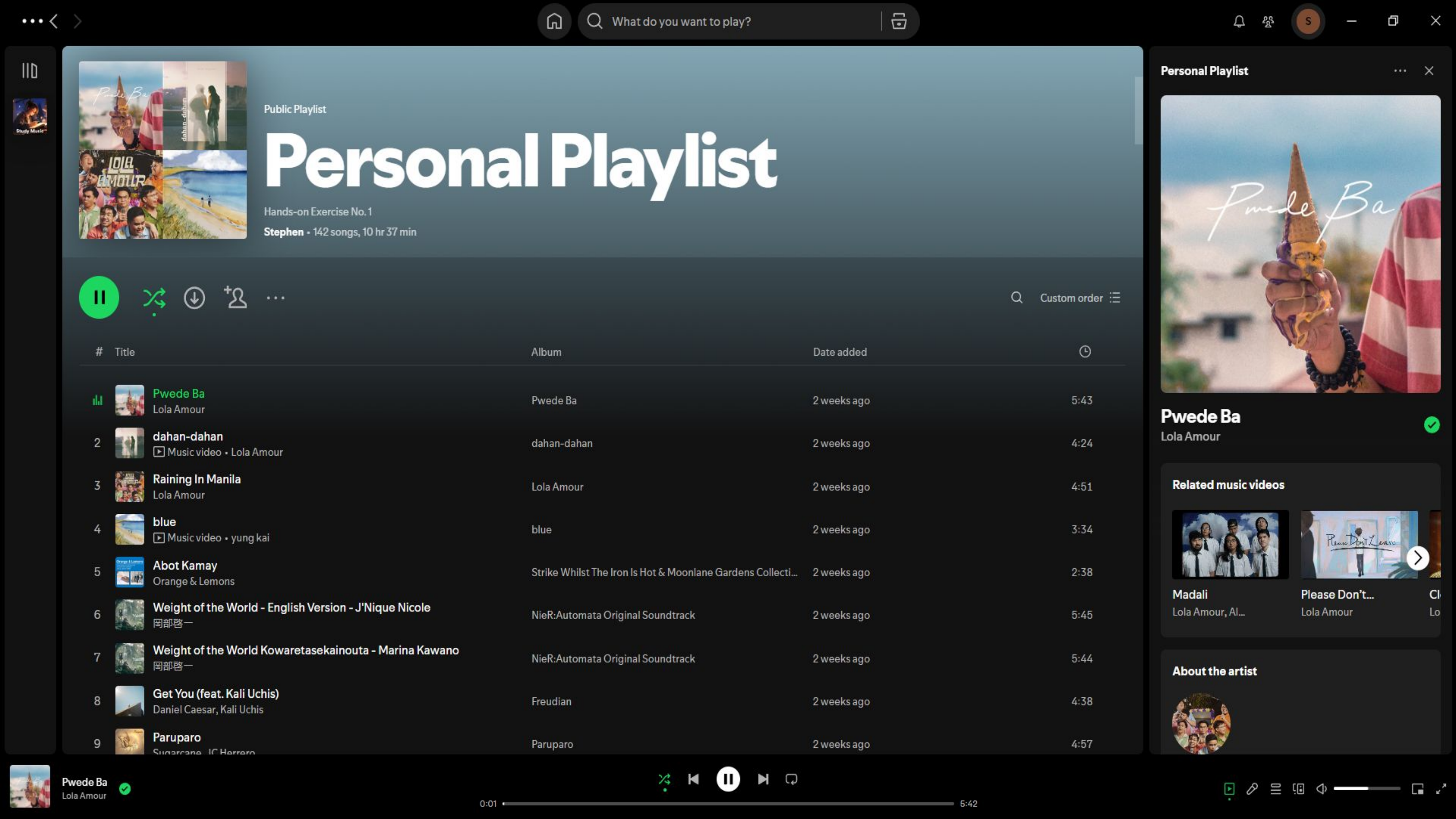
```
text = “I am an igloo”
example_set = {char for char in text if char not in “.!@#$%^&*()”}
```

Dict Comprehension

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

```
students = [“Gerry”, “Stewart”, “Oslo”]
example_dict = dict()
for student in students:
     students[student] = len(student)
```

```
students = [“Gerry”, “Stewart”, “Oslo”]
example_dict = { student: len(student) for student in students}
```



Public Playlist

Personal Playlist

Hands-on Exercise No. 1

Stephen • 142 songs, 10 hr 37 min

#	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

Pwede Ba
Lola Amour

Pwede Ba
Lola Amour
0:01 5:42

Personal Playlist

```
def add(song, playlist):
    # Add song to playlist
def remove(song, playlist):
    # Remove song from playlist
def play(playlist):
    # Print the first song in the playlist (if any) and remove
def show_all(playlist):
    # Print all contents in the playlist
def playlist_app():
    # Ask user what command they want to do

playlist_app()
```

H1

Ticket Management

Keeping track of multiple items

Ticket System

```python
def issue_ticket(ticket, queue):
    # Add a ticket to the queue
def process_ticket(queue):
    # Print and remove the first ticket in the queue (if any)
def cancel_ticket(ticket, queue):
    # Remove a specific ticket from the queue if any
def show_tickets(queue):
    # Print all the tickets in the queue
def ticket_app():
    # Ask user what command they want to do

playlist_app()
```

Challenge: Add Formatting

Challenge: Include other fields

Ticket System - Main Function

```
def ticket_app():
    # Initial Queue (You can add starting tickets)
    queue = []

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

    # Let user select action to do
    if user_choice == “add”:
        ticket = input(“Enter ticket name: “)
        issue_ticket(ticket, queue)
    elif ...

ticket_app()
```

02

Packaging

How to handle Python files properly

Modules and Packages

Module

Single Python file

```
.
└── module.py
```

Package

Folder with an `__init__.py`

```
.
└── package/
    ├── __init__.py
    └── module.py
```

Import Syntax

Accessing code outside of the current file

Basic Import

./hello.py

```python
def say_hello():
    print("Hello from module hello")
greeting = “Yellow!”
```

./main.py

```python
import hello

hello.say_hello()
```

```
.
├── hello.py
└── main.py
```

Specific Import

./hello.py

```python
def say_hello():
    print("Hello from module hello")
greeting = “Yellow!”
```

./main.py

```python
from hello import say_hello

say_hello()
```

```
.
├── hello.py
└── main.py
```

Basic Import with Alias

./hello.py

```python
def say_hello():
    print("Hello from module hello")
greeting = “Yellow!”
```

./main.py

```python
import hello as ho

ho.say_hello()
```

```
.
├── hello.py
└── main.py
```

Multiple Specific Import

./hello.py

```python
def say_hello():
     print("Hello from module hello")
greeting = “Yellow!”
```

./main.py

```python
from hello import say_hello, greeting

say_hello()
print(greeting)
```

```
.
├── hello.py
└── main.py
```

Basic Nested Import

./package/bye.py

```python
def say_bye():
    print("Goodbye from module bye")
```

./main.py

```python
import package.bye

package.bye.say_goodbye()
```

```
.
├── package
│    ├── __init__.py
│    └── bye.py
├── hello.py
└── main.py
```

Specific Nested Import

./package/bye.py

```python
def say_bye():
    print("Goodbye from module bye")
```

./main.py

```python
from package.bye import say_bye

package.bye.say_bye()
```

```
.
├── package
│   ├── __init__.py
│   └── bye.py
├── hello.py
└── main.py
```

Nested Import with Alias

./package/bye.py

```python
def say_bye():
    print("Goodbye from module bye")
```

./main.py

```python
import package.bye as pb

pb.say_bye()
```

```
.
├── package
│   ├── __init__.py
│   └── bye.py
├── hello.py
└── main.py
```

Standard Packaging Format

Most Python projects follow this project structure:

```
project_name/
   ├── LICENSE
   ├── pyproject.toml
   ├── README.md
   ├── src/
   │    ├── example_package_1/
   │    │    ├── __init__.py
   │    │    └── example.py
   │    └── example_package_2/
   │         ├── __init__.py
   │         └── example.py
   ├── tests/
   ├── doc/
   └── script/
```

Library Demo

A preview of the Python Standard Library

Try these Built-in Libraries!

Math

Common math constants and operations

Datetime

Dedicated package for handling calendar dates

Collections

Additional data structures

Time

Access to system time, delays, and conversions

SQlite

Quick setup for a light database system

Itertools

Efficient looping and combinatorials

Math Library

```python
import math

# Given radius
radius = 5

# Calculate area of the circle: A = π * r^2
area = math.pi * math.pow(radius, 2)

# Calculate circumference (perimeter) of the circle: C = 2 * π * r
circumference = 2 * math.pi * radius

# Calculate volume of the sphere: V = (4/3) * π * r^3
volume = (4/3) * math.pi * math.pow(radius, 3)
```

Itertools Library

```python
import itertools

items = ['A', 'B', 'C']

perm = itertools.permutations(items, 2)
print(list(perm))

comb = itertools.combinations(items, 2)
print(list(comb))
```

```
[('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B')]
[('A', 'B'), ('A', 'C'), ('B', 'C')]
```

Time Library

```python
import time

start_time = time.time()

for i in range(1000000):
    pass

end_time = time.time()

print("Time taken:", end_time - start_time, " seconds")
```

External Packages

Utilizing code from an separate source

Pip Install

The most common and straightforward way to install libraries and packages is through pip. It’s a built-in tool that comes with most Python distributions.

```
$ pip install package_name
```

Here is how to install one of the most downloaded Python packages for handling HTTP (Hypertext Transfer Protocol) - the common communication format in the web.

```
$ pip install requests
```

Virtual Environment

A virtual environment (venv) isolates packages for your project from the entire system. This prevents package conflicts, prevents clutter, and makes the project reproducible. The following code creates a folder .venv that will store isolated packages

Windows

```
$ python -m venv .venv
```

Linux/MacOS

```
$ python3 -m venv .venv
```

Virtual Environment - Activation

To actually use the packages of a virtual environment, you need to **`activate`** it first.

Windows (Command Prompt)

```
$ .venv\Scripts\activate
```

Windows (Powershell)

```
$ .venv\Scripts\Activate.ps1
```

Linux/MacOS

```
$ source .venv/bin/activate
```

Virtual Environment - Deactivation

To exit the virtual environment, simply enter **`deactivate`** on any console

```
$ deactivate
```

Virtual Environment - Requirements

The installed packages (and their accepted versions) can be saved with the following commands. The `.venv` folder doesn’t need to be copied if you can get a list of required packages to make the project work and install it at the start.

Writing Requirements

```
$ pip freeze > requirements.txt
```

Installing requirements

```
$ pip install -r requirements.txt
```

F3

Request Library

Getting information online programmatically

Request Library Demo

This codes gets the contents of the BBC news page

```
import requests

url = "https://www.bbc.com/news"
response = requests.get(url)

if response.status_code == 200:
     print(response.text)

else:
    print(f"Failed! Status code: {response.status_code}")
```

H3

Code Organize

Simple exercise to consider how to separate code

```python
def calculate_area_circle(radius):
    return 3.14 * radius * radius
def calculate_area_rectangle(length, width):
    return length * width
def calculate_area_triangle(base, height):
    return 0.5 * base * height

def main():
    inputs = {“radius”: 5, “length”: 10, “width”: 4, “base”: 8, “height”: 6}

    circle_area = calculate_area_circle(inputs[“radius”])
    rectangle_area = calculate_area_rectangle(inputs[“length”], inputs[“width”])
    triangle_area = calculate_area_triangle(inputs[“base”], inputs[“height”])

    print("Circle area:", circle_area)
    print("Rectangle area:", rectangle_area)
    print("Triangle area:", triangle_area)

main()
```

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

```
message = """
Hello World
Hello World
Hello World
"""
```

Result in Console:

```

Hello World
Hello World
Hello World
```

Docstrings

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

```python
def helpful_function():
    """
    Adding a multiline string after a function definition
    creates a guide when calling the help function
    """
    return 0

help(helpful_function)
```

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)
```

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

```python
number = 1.123456789
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:,}"

```python
number = 123456789
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 commas

```python
number = 123456.789
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

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

Result in Console:

```
98.99%
```

Quick Exercise: Simple Interest

Ask the user for the information of three items

```
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)
```

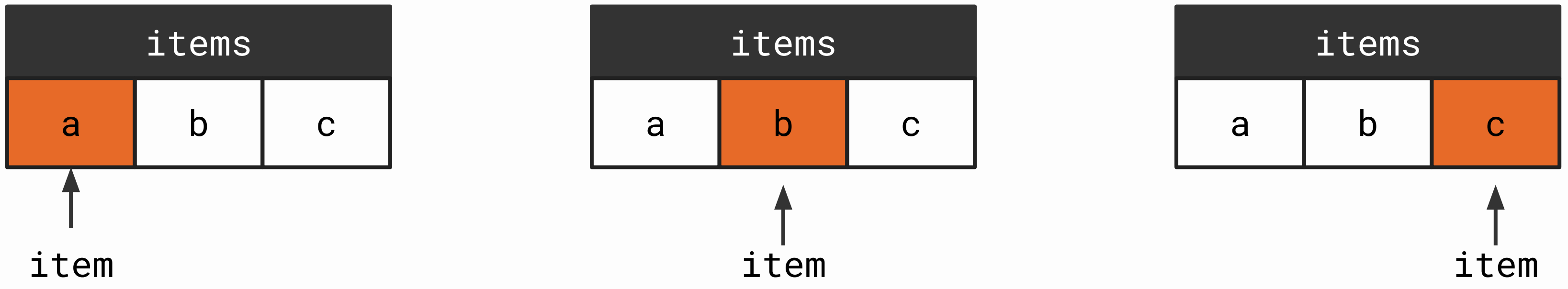
Char Sequence

Strings are a sequence of letters

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)
```



Quick Exercise: Secret Code

Given the following string

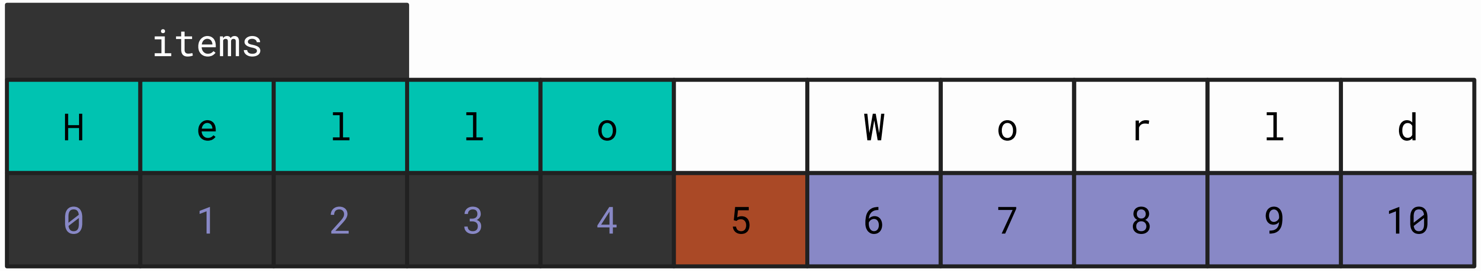
```python
text = "qxwuxaisdecktbelxjy"
```

Find the secret word by getting every third letter starting from **“q”**

Substrings

Strings also support indexing and slicing access (not modification)

```
items = ‘Hello World’
print(items[:5])
```



Quick Exercise: Secret Code (Quickly)

Given the following string

```
text = "qxwuxaisdecktbelxjy"
```

Find the secret word by getting every third letter starting from **“q”**

Substring Finding

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

```
message = ‘Hello World’
print(‘World’ in message)
```

```
True
```

Case Change

Applying formatting to an entire string

String Lowercase

The **`lower`** method returns a lowercase copy of the string.

```
example = “Hello World”
```

```
var_example = example.lower()
print(example)
print(var_example)
```

```
Hello World
hello world
```

String Uppercase

The **`upper`** method returns a uppercase copy of the string.

```
example = “Hello World”
```

```
var_example = example.upper()
print(example)
print(var_example)
```

```
Hello World
HELLO WORLD
```

String Title Case

The **`title`** method returns a copy of the string where every first letter is capitalized

```
example = “This is a title”
```

```
var_example = example.title()
print(example)
print(var_example)
```

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

Use Case: Sanitized User Input

A very common use for the **`upper`** or **`lower`** method is to standardize cases

```
user_input = input(“Proceed (Yes/yes/y)? ”)
if user_input == “Yes” or user_input == “yes”:
    print(“Proceeding”)
```

```
user_input = input(“Proceed (Yes/yes/y)? ”)
if user_input.lower() == “yes”:
    print(“Proceeding”)
```

Quick Exercise: Angery

Given a regular string input

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

Convert it to an angrier version:

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

Case Check

Checking string formatting

String Check Lowercase

The **`islower`** method returns **`True`** if all the characters are in lowercase

```
example = “hello”
```

```
all_lower = example.islower()
print(example)
print(all_lower)
```

```
hello
True
```

String Check Uppercase

The **`isupper`** method returns **`True`** if all the characters are in uppercase

```
example = “HELLO”
```

```
all_upper = example.isupper()
print(example)
print(all_upper)
```

```
HELLO
True
```

String Check Space

The **isspace** method returns **True** if all the characters are just spaces

```
example = “     ”
```

```
all_space = example.isupper()
print(example)
print(all_space)
```

```

True
```

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

The **isalpha** method returns **True** if all characters are part of a human writing system

```
example = “すごい”
```

```
all_alpha = example.isalpha()
print(example)
print(all_alpha)
```

```
すごい
True
```

String Check Numeric

The **`isdigit`** method returns `True` if all characters are digits

```
example = “12345”
```

```
all_numeric = example.isnumeric()
print(example)
print(all_numeric )
```

```
12345
True
```

Quick Exercise: Number Check

Ask the user for an input

```
user_input = input(“Please provide a positive number: “)
```

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

```
This is a valid number
```

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

String Edge

Check the start or end of a string

String Check Prefix

The **`startswith`** method returns **`True`** if the substring is a subset starting from the first char

```
example = “Hello World”
```

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

```
Hello World
True
```

String Check Suffix

The **endswith** method returns **True** if the substring is a subset by the final char

```
example = “Hello World”
```

```
worldly = example.endswith(“World”)
print(example)
print(worldly)
```

```
Hello World
True
```

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.
```

Word Handling

Common string methods to handle complex formatting issues

String Strip

The **`strip`** method returns a copy of the string without spaces in its left and right

```
example = “        Hello World                 ”
```

```
clean_example = example.strip()
print(example)
print(clean_example)
```

```
    Hello World
Hello World
```

Use Case: Sanitized User Input

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

```
user_input = input(“Proceed (Yes/yes/y)? ”)
clean_input = user_input.lower().strip()
if clean_input == “yes”:
    print(“Proceeding”)
```

String Replace

The **`replace`** method returns a copy of the string but the given substring replaces every instance of the second substring given

```
example = “123,456,789”
```

```
alternative_example = example.replace(‘,’, ‘.’)
print(example)
print(alternative_example)
```

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

String Replace to Remove

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

```
example = “a, b, c, d”
```

```
alternative_example = example.replace(“, “, ““)
print(example)
print(alternative_example)
```

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

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
```

String Split

The **`split`** method returns a list of words found in the string (determined using spaces)

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

```
words = example.split()
print(example)
print(words)
```

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

String Join

The **join** method returns the concatenation of a list of strings, glued using the string

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

```
combined_words = ” “.join(example)
print(example)
print(combined_words)
```

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

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
```

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

```python
import re

text = "Call me at 123-456-7890"
numbers = re.findall(r"\d+", text)
print(numbers)
```

Quick Exercise: Crucial Dates

Given the following string

```
"The event is on 12/15/2023, and the deadline is 01/01/2024."
```

Print all of the dates mentioned

```
[“12/15/2023”, “01/01/2024”]
```

Regex Replace

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

```python
import re

text = "Alice has an apple and an avocado."
pattern = r"\ba\w*"
result = re.sub(pattern, "X", text)

print(result)
```

Quick Exercise: Fruit Swap

Given the following string

```
"I like apple pie; apple is my favorite fruit."
```

Replace every instance of “apple” with “buko”

```
"I like buko pie; buko is my favorite fruit."
```

F3

Web Scraping Demo

Processing complex strings

Text Processing - Manual

```python
import requests
import re

def decode_html_entities(text):
      entities = { "&#x27;": "'", "&quot;": '"', "&amp;": "&", "&lt;": "<", "&gt;": ">"}
      for entity, char in entities.items():
            text = text.replace(entity, char )

      return text

url = "https://www.bbc.com/news"
response = requests.get(url)

if response.status_code == 200:
      cleaned_html = re.sub(r"<!--.*?-->", "", response.text)
      headlines = re.findall(r"<h2.*?>(.*?)</h2>", cleaned_html)

      for headline in headlines:
            decoded_headline = decode_html_entities(headlines).strip()
            print(decoded_headline )
else:
    print(f"Failed! Status code: {response.status_code}")
```

Text Processing - Beautiful Soup

```python
import requests
from bs4 import BeautifulSoup

url = "https://www.bbc.com/news"
response = requests.get(url)

if response.status_code == 200:
    soup = BeautifulSoup(response.text, "html.parser")
    headlines = soup.find_all("h2")

    for headline in headlines:
        print(headline.get_text())
else:
    print(f"Failed! Status code: {response.status_code}")
```

H3

Business Reporting

Using string handling techniques to format fixed data

Business Report - Given Data

```
{
    "company_name": "TechX Innovations",
    "report_period": "Q1 2024",
    "sales_data": [
        {
            "month":"jAnuarY","revenue":”25,000”,"region":"America"
        },
        {
            "month":"February","revenue":”30 000”,"region":"EUROPE"},
        {
            "month":"march","revenue":”28000”,"region":"Asia-Pacific"
        }
    ]
}
```

Business Report - Expected Format

```
Company: TechX Innovations
Report Period: Quarter 1, 2024

January: Revenue = $25,000, Region = North America
February: Revenue = $30,000, Region = Europe
March: Revenue = $28,000, Region = Asia-Pacific
```

04

File Handling

More permanent approach to data

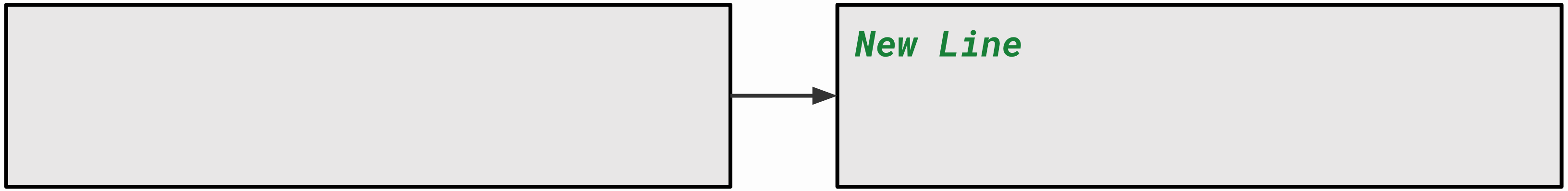
Text Files

The most common and well-known file type

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()*

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



Quick Exercise: Write Guestlist

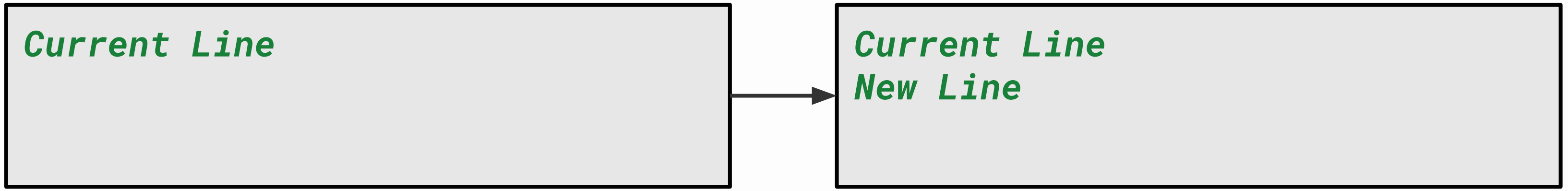
The boss has given you a list of attendees that are allowed in the event. Write them in a file:

```
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()**

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



```
Current Line
```

```
Current Line
New Line
```

Quick Exercise: Append Guestlist

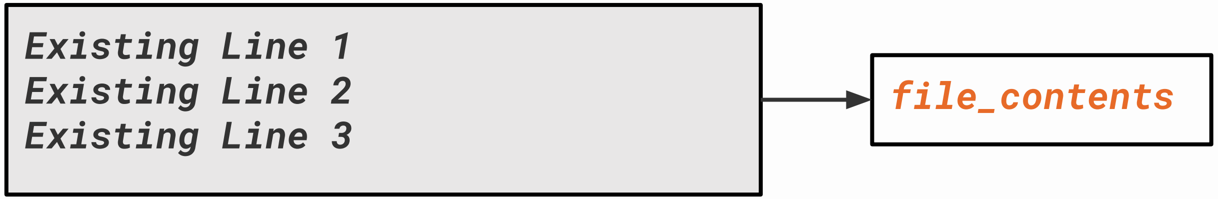
The boss forgot to add herself. Add her name in the last part

```
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()`***

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

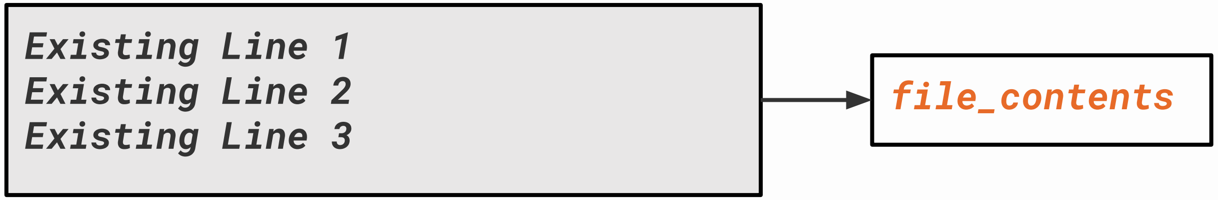


```
Existing Line 1
Existing Line 2
Existing Line 3
```

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()`***

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



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

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.

```json
{
  "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.

```python
import json

data = [
    {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
    {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
]

with open(‘people.json’, ‘w’) as file:
    json.dump(data, file)
```

JSON Dump (Formatted)

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

```
import json

data = [
     {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
     {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
]

with open(‘people.json’, 'w’) as file:
     json.dump(data, file, indent=4)
```

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": 40, "Description": "Fresh apple"},
    {"Name": "Rice",   "Price": 60, "Description": "A kilo of rice."},
]
```

JSON Load

JSON handling requires importing a built-in module

```
import json

with open(‘people.json’, 'r’) as file:
    data = json.load(file)

print(data)
```

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 represents tabular data with a column header that's similar to Microsoft Excel Sheets and Google Sheets.

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

CSV Writing (with Lists)

```python
import csv

data = [
     ['Name', 'Age', 'Occupation'],
     ['Alice', 30, 'Engineer'],
     ['Bob', 25, 'Designer'],
]

with open(‘people.csv’, 'w', newline='') as file:
     writer = csv.writer(file)
     writer.writerows(data)
```

```
['Alice', 30, ‘Engineer’]
```

$\rightarrow$

```
Alice, 30, Engineer
```

CSV Writing (with Dicts)

```python
import csv

data = [
     {'Name': 'Alice', 'Age': 30, 'Occupation': 'Engineer'},
     {'Name': 'Bob', 'Age': 25, 'Occupation': 'Designer'},
]

with open(‘people.csv’, 'w', newline=’’) as file:
     writer = csv.DictWriter(file, fieldnames=data[0].keys())
     writer.writeheader()
     writer.writerows(data)
```

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

```
Alice, 30, Engineer
```

Quick Exercise: Employee Database

Create a csv with the given table:

Name	Employee ID	Role
Alice	123	Engineer
Bob	456	Designer
Charlie	789	Lawyer
Delta	992	Accountant

CSV Reading (as Lists)

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

```
import csv

with open(‘people.csv’, 'r', newline='') as file:
    reader = csv.reader(file)

    for row in reader:
        print(row)
```

CSV Reading (as Dicts)

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

```
import csv

with open(‘people.csv’, 'r', newline='') as file:
    reader = csv.DictReader(file)

    for row in reader:
        print(row)
```

Quick Exercise: Employee Check

Reload the entries saved in the previous CSV file and print it in this format:

```
Employees
 1. Alice    [123]: Engineer
 2. Bob      [456]: Designer
 3. Charlie  [789]: Lawyer
 4. Delta    [992]: Accountant
```

F4

Console Notepad

Quick recap of file read and write

Console Notepad

```
def save_note(filename, content):
    """Saves the content to a file."""

def read_note(filename):
    """Reads and returns the content of a file."""

def main():
    """Simple interface to save and read notes."""

main()
```

H4

Ticket Storage

Keeping track of multiple items and saving for later

Ticket Storage

```
def issue_ticket(ticket, queue): ...
def process_ticket(queue): ...
def cancel_ticket(ticket, queue): ...
def show_tickets(queue): ...
def ticket_app(): ...

def save(filepath, queue): ...
def load(filepath): ...

playlist_app()
```

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"],
}
```

War: Two-Player Game of Luck

Here is the standard rules of War

- A standard deck is face-down (not visible), shuffled and evenly split between two players
- The game continues until one player has all of the cards. For every round:
    - Both players flip their top card.
    1. The player with the higher card wins both and adds them to their deck.
1. If the cards are equal, it’s WAR:
1. Each player puts down 3 cards face-down and a 4th face-up.
2. The higher face-up card wins all 10 cards.
    - Repeat if tied again.
    -

The game continues until one player has all the cards.

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.

```
from random import choice
```

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

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

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
```

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

Intermediate Python