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  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2  print(ranks)
```

ranks												
A	2	3	4	5	6	7	8	9	10	J	Q	K

List Looping

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

```python
ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
for rank in ranks:
    print(rank)
```

```
First Message
Second Message
Third Message
```
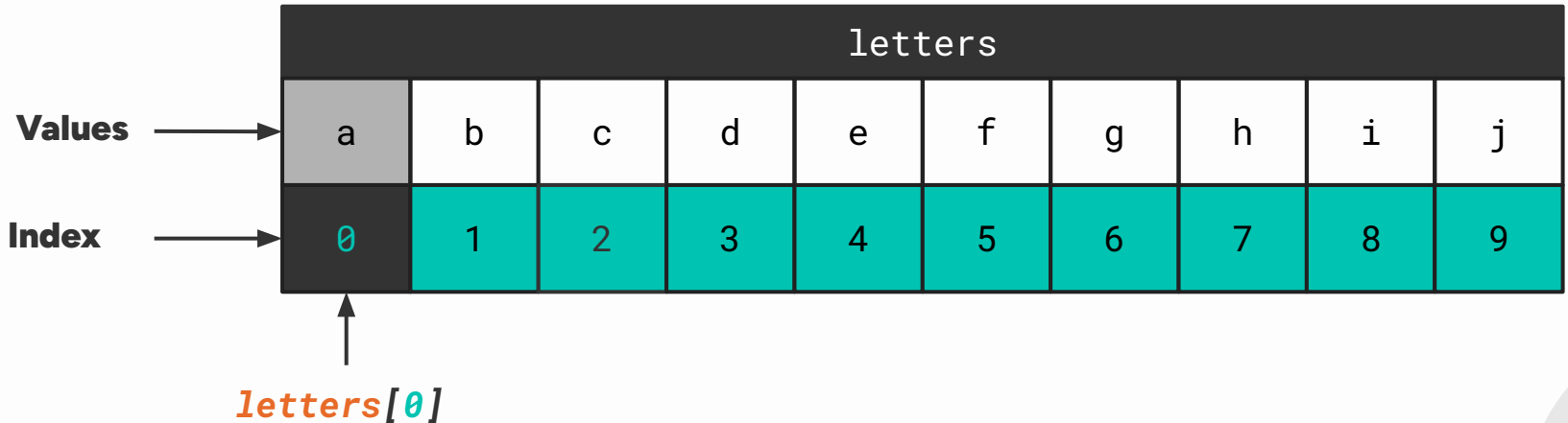
# Index Logic

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

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

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

Values

Index

letters[5]

Item Access

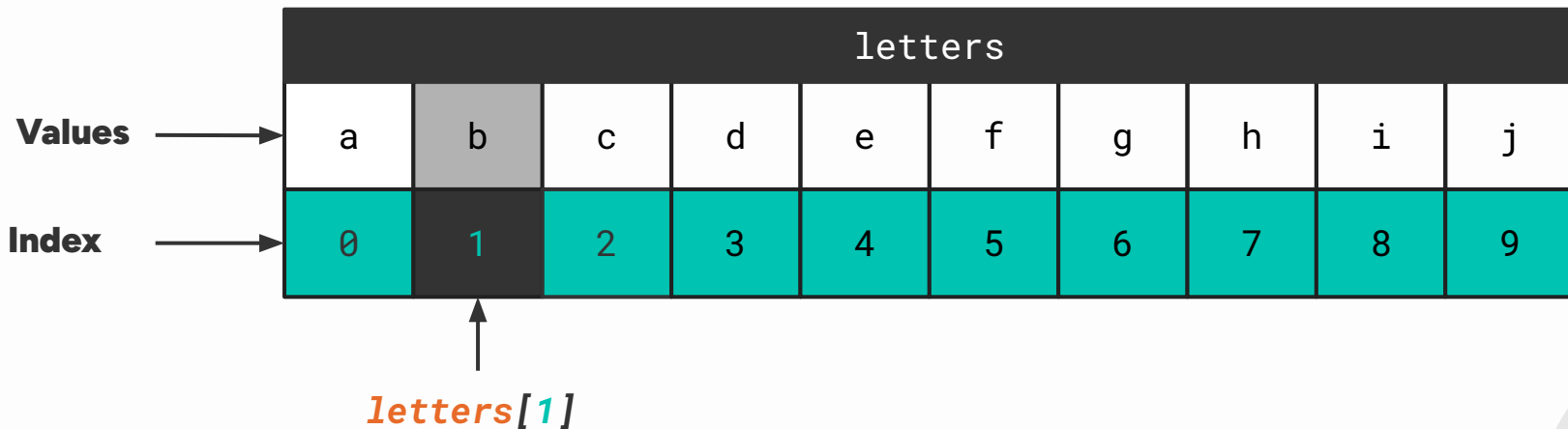Specific values can be accessed in a list by using the list name, square brackets, and index.

```
1  letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2  print(letters[0])
```
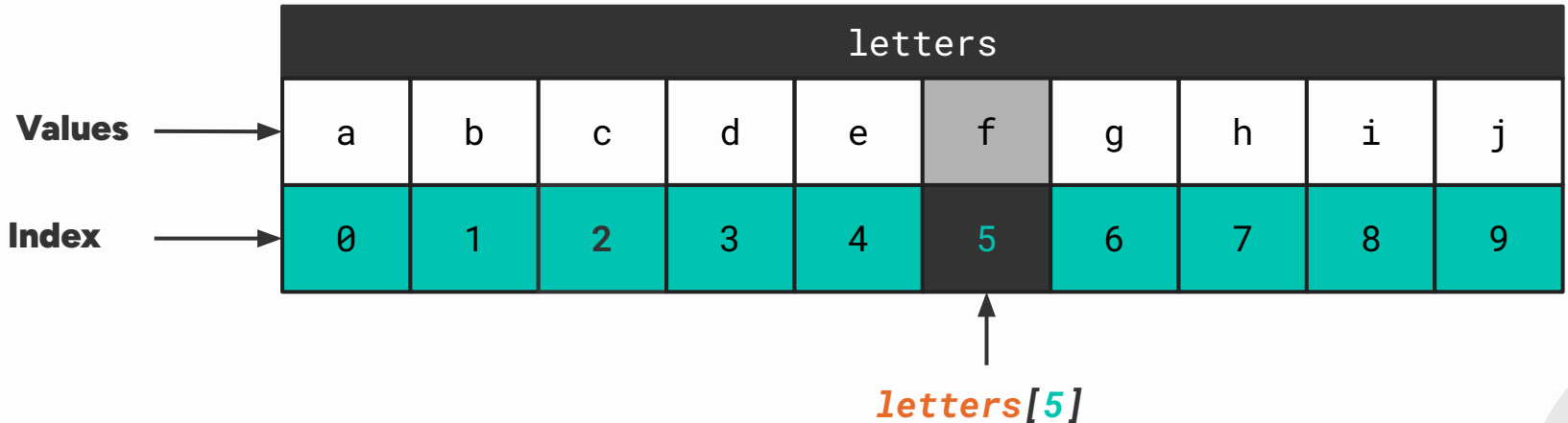
letters									
a	b	c	d	e	f	g	h	i	j
0	1	2	3	4	5	6	7	8	9

Values →

Index →

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])
```

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

Values →

Index →

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])
```

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

Values →

Index →

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])
```

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

Values →

Index (+) →

Index (-) →

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									
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

Values →

Index (+) →

Index (-) →

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 [0]

letters [-12]

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 [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

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 [6]
letters [-6]

Quick Exercise: Royal Flush

01_royal_flush.py

```python
ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']

# Print '10','J','Q','K',and 'A'  from list
print()
```

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] = '*'
```
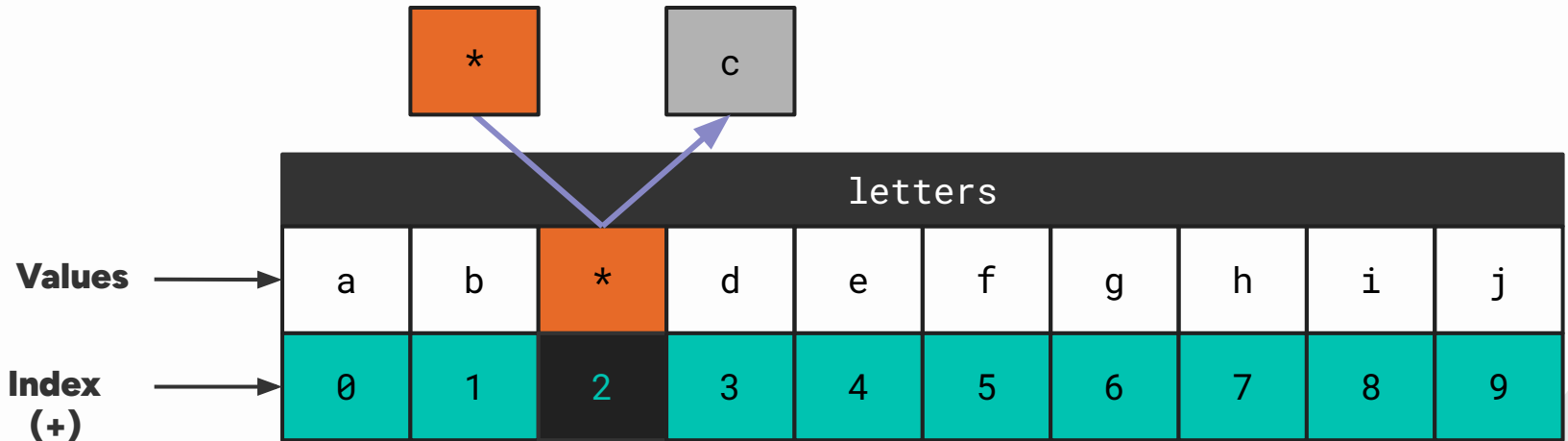
Quick Exercise: Royal Draw

```python
02_royal_draw.py
1  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2
3  # Change '10','J','Q','K',and 'A' cards to 'Drawn'
4
5  # Show the changed list
6  print(ranks)
```

Tuple Definition

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

```
1  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2  print(ranks)
```

ranks												
A	2	3	4	5	6	7	8	9	10	J	Q	K

Tuple Modification

Tuples cannot modify its contents after creation

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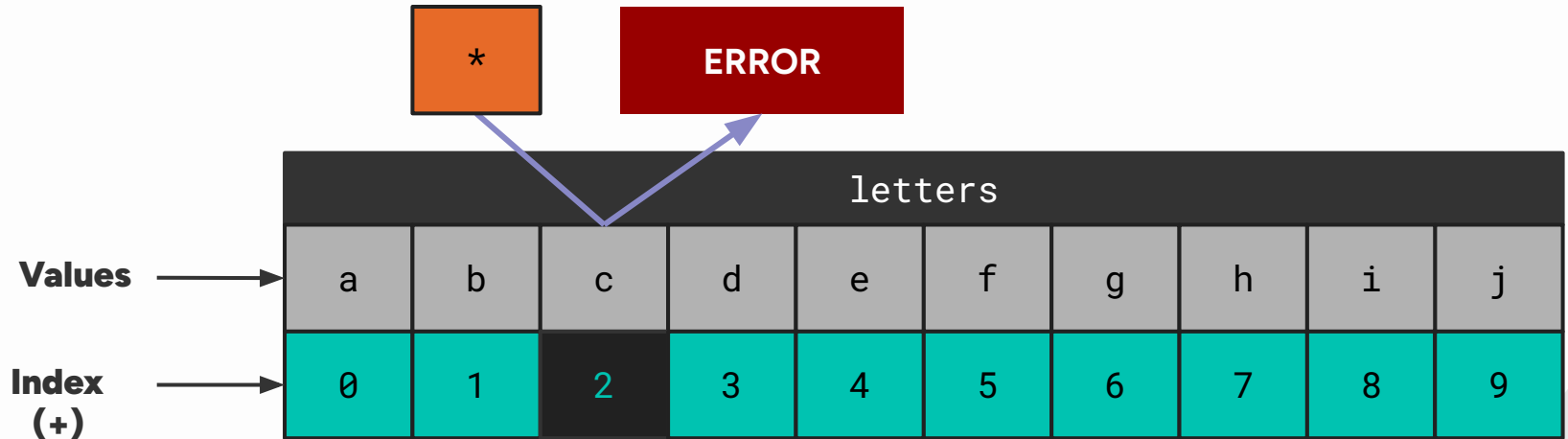
Nested Data

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)]
```

student_data[0][0]

Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

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

student_data[0][0]

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)]
```

student_data*[0][0]*

student_data*[0][1]*

student_data*[1][0]*

student_data*[1][1]*

student_data*[2][0]*

student_data*[2][1]*

NASAAN KA NA

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 [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			
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 [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

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 **[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)
```

items
a	b	c

↑
item

items
a	b	c

↑
item

items
a	b	c

↑
item
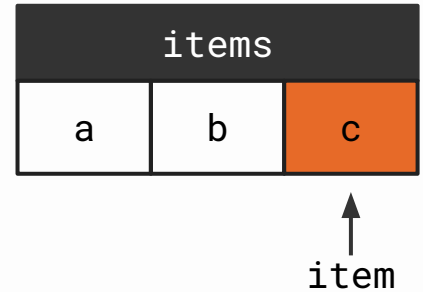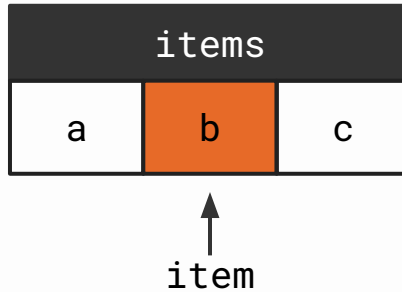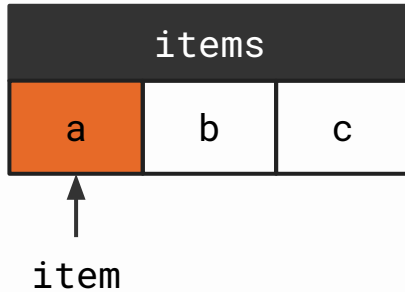
Multiple Looping

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

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

a	b	c

1	2	3

↑
item, other

a	b	c

1	2	3

↑
item, other

a	b	c

1	2	3
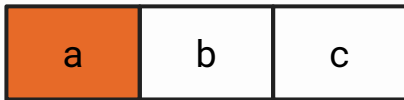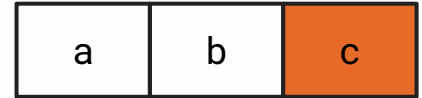
↑
item, other

Multiple Loopings Example

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

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

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

```
| 1      | Google    | 10000  PHP |
| 2      | Jollibee  | 20000  PHP |
| 3      | Nvidia    | 3000   PHP |
```

Quick Exercise: Student Records

03_student_records.py

```python
1  student_names = ("Juan", "Maria", "Joseph")
2  student_scores = (70, 90, 81)
3
4  """
5  Print the student scores and names in the following format
6  Student Records:
7      Student: Juan scored 70 in the exam.
8      Student: Maria scored 90 in the exam.
9      Student: Joseph scored 81 in the exam.
10 """
11 print(f"Student: name scored score in the exam")
```

Challenge: Print the highest scorer

Enumerate Looping

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

```
1  names = ('Jeff', 'Alex', 'Kim')
2  for index, name in enumerate(names):
3      print(index, name)
```

```
0 Jeff
1 Alex
2 Kim
```

Enumerate Looping (Different Start)

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

```python
names = ('Jeff', 'Alex', 'Kim')
for index, name in enumerate(names, start=1):
    print(index, name)
```

```
1 Jeff
2 Alex
3 Kim
```

Quick Exercise: Inventory Check

04_inventory_check.py

```python
inventory = ("Mousepad", "Keyboard", "Monitor", "Cable")

"""
Print the items in the inventory and the order they appear:
    Item 1: Mousepad
    Item 2: Keyboard
    Item 3: Monitor
    Item 4: Cable
"""
```

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)
```
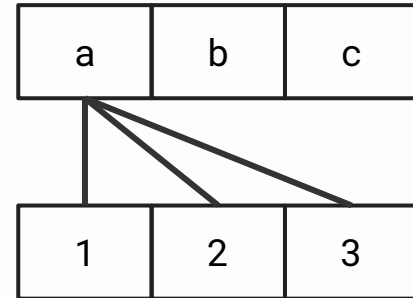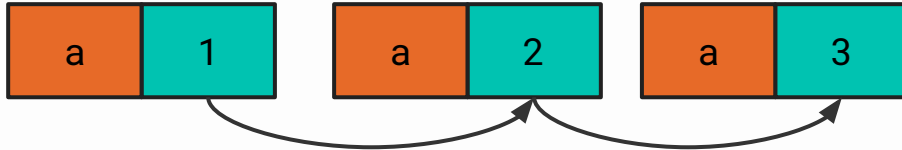
Nested Looping

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)
```

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: Standard Deck

```
05_standard_deck.py
```

```python
ranks = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
suits = ("Hearts", "Diamonds", "Clubs", "Spades")
"""
Print every possible pairing of ranks and suits
A of Hearts
2 of Hearts
3 of Hearts
...
K of Hearts
A of Diamonds
2 of Diamonds
3 of Diamonds
...
"""
```

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]
```

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

['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: Royal Flush (version 2)

```python
01_royal_flush.py
1  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2
3  # Print '10','J','Q','K', and 'A' from ranks
4  print()
```

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: Second Draw

06_second_draw.py

```python
ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']

# Draw every other card
print(ranks)
```

Operations

Applicable operations for lists and tuples

Addition

Two or more lists or tuples can be combined into a new, singular list or tuple

```
1  numbers_cards = ["1","2","3","4","5","6","7","8","9"]
2  special_cards = ["+2","skip","reverse"]
3  super_cards = ["0","+4","color"]
4
5  cards = numbers_cards + special_cards + super_cards
6
7  print(cards)
```

Multiplication

Similar to strings, lists and tuples can also be multiplied

```
1  numbers_cards = ["1","2","3","4","5","6","7","8","9"]
2  special_cards = ["+2","skip","reverse"]
3  super_cards = ["0","+4","color"]
4
5  max_cards = 8 * (special_cards + numbers_cards)
6  min_cards = 4 * super_cards
7
8  print(max_cards + min_cards)
```

Quick Exercise: Funny Binary

```python
07_funny_binary.py
```

```python
# Create the binary for letter 'h' as a list of 1's and 0's
binary_h = list(bin(ord('h')))
binary_h = binary_h[2:]

# Create the binary for letter 'a' as a list of 1's and 0's
binary_a = list(bin(ord('a')))
binary_a = binary_a[2:]

# Create the binary for 'hahaha'
binary = []
print(binary)
```

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

```python
1  response = input("Proceed: ")
2  if response == "Yes" or response == "yes" or response =="y":
3      print("Proceeding")
```

This is an equivalent statement

```python
1  response = input("Proceed: ")
2  if response in ("Yes","yes","y"):
3      print("Proceeding")
```

Quick Exercise: Banned

```
08_banned.py
1  banned_words = ("moist", "break", "raise")
2
3  # Ask the user for a word
4  # If the word is in banned_words, say "Banned"
5
6  print("Banned")
```

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]
```

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]
```

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]
```

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]
```

Quick Exercise: Class Statistics

09_class_statistics.py

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

# Print the average score
average_score = None
print(average_score)
```

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]
```

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]
```

Quick Exercise: Class Statistics (v2)

09_class_statistics.py

```python
1  student_scores = [98, 75, 100, 86, 100, 3]
2
3  # Print the average score
4  average_score = None
5  print(average_score)
6
7  # Print the rankings, highest to lowest
8  print()
```

Methods

Modifying the function directly

Functions vs Methods

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

```
value.method()
```

function

input

[1, 2, 3]

example

[1, 2, 3]

[1+1, 2+1, 3+1]

[2, 3, 4]

method

input

[1, 2, 3]

example

[1, 2, 3]

[1+1, 2+1, 3+1]

example

[2, 3, 4]
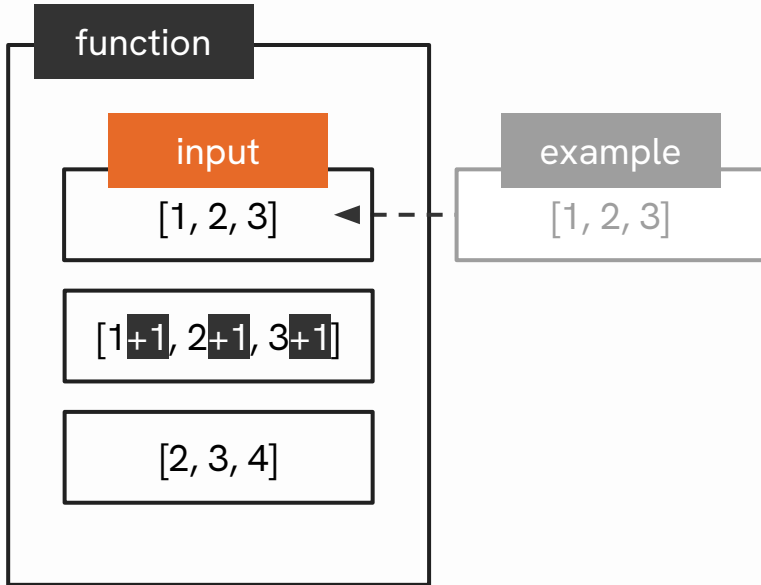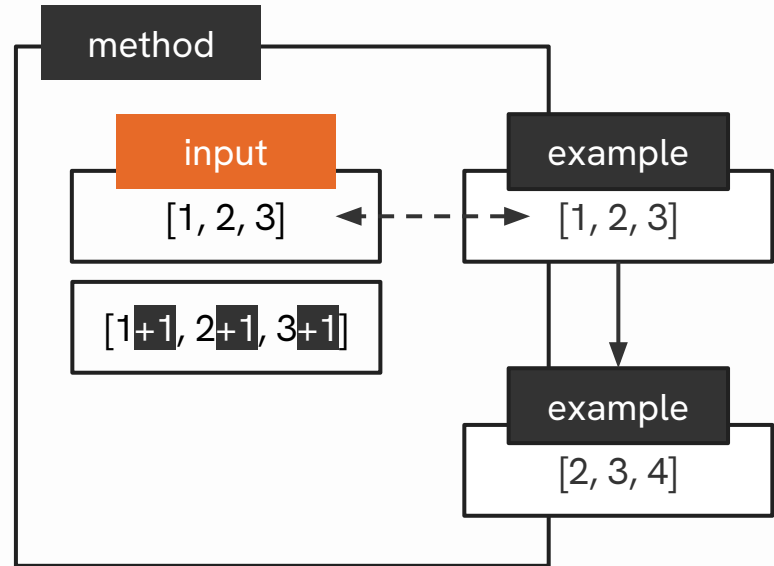
Methods Affects Readable Data

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

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

print(data)
function(param=data)
print(data)
```

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]
```

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]
```

Quick Exercise: Attendance

```
10_attendance.py
```

```python
1  attendee_names = []
2
3  attendee_count = int(input("Attendee count: "))
4
5  # Do this for as many attendees expected
6  attendee_name = input("Attendee name: ")
7  # Add attendee_name to attendee_names
8
9  print(attendee_names)
```

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]
```

Safe Remove Method

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

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

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

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

Quick Exercise: Attendance (v2)

```
10_attendance.py
1  attendee_names = []
2
3  attendee_count = int(input("Attendee count: "))
4
5  # Do this for as many attendees expected
6  attendee_name = input("Attendee name: ")
7  # Add attendee_name to attendee_names
8
9  print(attendee_names)
10
11 # If your name is in attendee_names, remove it
12 print(attendee_names)
```

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]
```

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]
```

Quick Exercise: Attendance (v3)

10_attendance.py

```python
attendee_names = []

attendee_count = int(input("Attendee count: "))

# Do this for as many attendees expected
attendee_name = input("Attendee name: ")
# Add attendee_name to attendee_names

print(attendee_names)

# If your name is in attendee_names, remove it
print(attendee_names)

# Remove and print the late attendee (last attendee)
```

Cost Tracker

Implementing your own stack

```python
# 11_cost_tracker.py

def spend(expenses, cost):
    """Add a new cost in the list of expenses"""
def refund(expenses):
    """Remove the last cost added (if any)"""
def show(expenses):
    """Print the current list of expenses line-by-line"""
```

```
11_cost_calculator.py

def main():
    current_expenses = []
    command = input("Command: ")

    while command:
        # Handle command here
        # Example: user inputs "add" -> do add(expenses, cost)
        # Ask the inputs here, not in the function

        # Ask for more
        command = input("Command: ")

main()
```

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

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}
```

Quick Exercise: Unique Attendance

```
12_unique_attendance.py
1  attendee_names = set()
2
3  attendee_count = int(input("Attendee count: "))
4
5  # Do this for as many attendees expected
6  attendee_name = input("Attendee name: ")
7  # Add attendee_name to attendee_names
8
9  print(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}
```

Quick Exercise: Unique Attendance (v2)

`12_unique_attendance.py`

```python
attendee_names = set()

attendee_count = int(input("Attendee count: "))

# Do this for as many attendees expected
attendee_name = input("Attendee name: ")
# Add attendee_name to attendee_names

# Remove your name from attendees (if there)

print(attendee_names)
```

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
```

Quick Exercise: Unique Attendance (v3)

```
12_unique_attendance.py
```

```python
1  attendee_names = set()
2
3  attendee_count = int(input("Attendee count: "))
4
5  # Do this for as many attendees expected
6  attendee_name = input("Attendee name: ")
7  # Add attendee_name to attendee_names
8
9  # Remove your name from attendees (if there)
10
11 print(attendee_names)
12
13 # Pick a random raffle winner (attendee)
```

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

Set1					
a	b	c	d	e	f
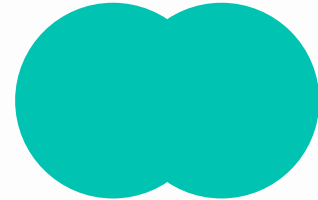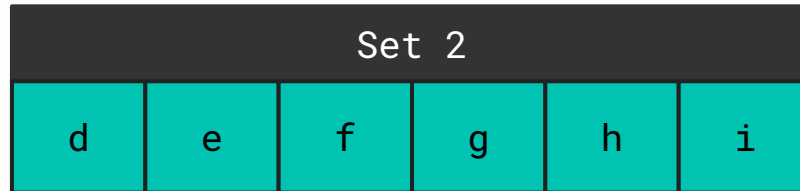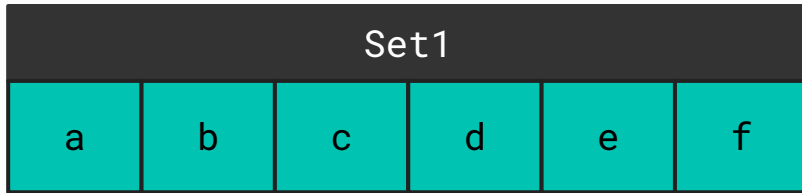
Set 2					
d	e	f	g	h	i

Set Intersection

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

Set1					
a	b	c	d	e	f

Set 2					
d	e	f	g	h	i

Set Difference

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

Set1					
a	b	c	d	e	f
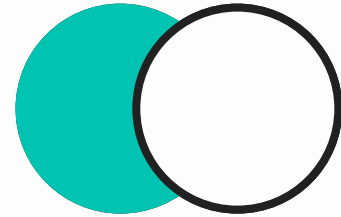
Set 2					
d	e	f	g	h	i

Set Difference (Order Matters)

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

Set1					
a	b	c	d	e	f

Set 2					
d	e	f	g	h	i
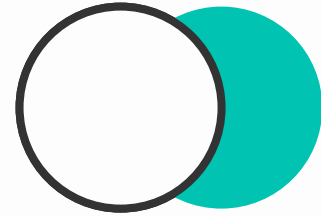
Set Symmetric Difference

```python
1  set1 = {'a', 'b', 'c', 'd', 'e', 'f'}
2  set2 = {'d', 'e', 'f', 'g', 'h', 'i'}
3  print(set1.symmetric_difference(set2))
4  print(set1 ^ set2)
```
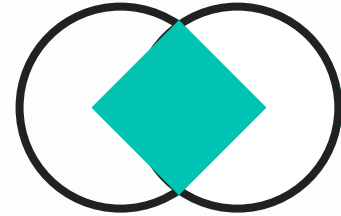
Set1					
a	b	c	d	e	f

Set 2					
d	e	f	g	h	i
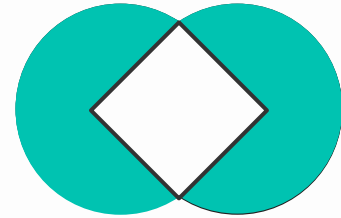
Quick Exercise: Gate Crashing

13_gate_crashers.py

```python
invited = {"Ana", "Ben", "Carlo", "Dani"}
attended = {"Ben", "Carlo", "Ely"}

# Who are all the involved members?
print("Involved Members:")

# Who was absent?
print("Absent:")

# Who gatecrashed?
print("Not enrolled but attended:")

# Who was invited and attended
print("Attended properly:")
```

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

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.

```
key  →  value
```

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

Example 01: Form Data

Dictionaries can be used to contain different data for one concept or group

```python
form_data = {
    "first_name": "Juan",
    "last_name": "Dela Cruz",
    "age": 25,
    "newsletter": True
}
```

Example 02: Conversion

It's common to convert one value to another (mapping) using dictionaries

```
status_codes = {
    200: "OK",
    404: "Not Found",
    500: "Server Error"
}
```

Quick Exercise: Country Codes

14_country_codes.py

```python
# Add more country codes
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

print(country_codes)
```

Dictionary Access

The dictionary values can be accessed using their keys, The syntax is the same as indexing with lists and tuples, but it uses keys instead of index. If it's not there, it raises a **KeyError**

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

print(student_records["Joseph"])
```

```
81
```

Quick Exercise: Country Codes (version 2)

```python
# Add more country codes
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

# Print the country for the given country code
country_code = input("Enter country code: ")
print(country_codes)
```

Dictionary Access (Safe)

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

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

print(student_records.get("Elizabeth"))
```

None

Dictionary Access (Safe)

The **get** method can also take an optional parameter that it returns if the key is not found

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

-1

Quick Exercise: Country Codes (version 3)

```python
# Add more country codes
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

# Print the country for the given country code
# If the key is not found, print Unknown
country_code = input("Enter country code: ")
print(country_codes)
```

Dictionary Loops

Handling a set and list at once

Dictionary Iteration (Keys)

The dictionary keys can be accessed using the keys method

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

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

Dictionary Iteration (Keys)

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

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

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

Quick Exercise: Country Codes (version 4)

```python
# Add more country codes
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

# Print the country for the given country code
# If the key is not found, print Unknown
country_code = input("Enter country code: ")
print(country_codes)

# Print all codes
```

Dictionary Iteration (Values)

The dictionary values can be accessed using the values method

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

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

Quick Exercise: Country Codes (version 5)

```python
# Add more country codes
country_codes = {
    "PH": "Philippines",
    "US": "United States",
}

# Print the country for the given country code
# If the key is not found, print Unknown
country_code = input("Enter country code: ")
print(country_codes)

# Print all codes

# Print all countries
```

Dictionary Iteration (Key-Value)

Both key and values can be accessed using the items method

```python
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: Wishlist

15_wishlist.py

```python
# Fill in the details of the item you plan to buy
item = {
    "Name": ...,
    "Info": ...,
}

# Print the item details in the following format:
"""
Item:
    Name: item name
    Info: item info
    ...
"""
```

Dictionary Add

Dictionaries are write-safe at a cost

Dictionary Entry

For dictionaries, adding and creating new entries is the same

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

25

Dictionary Overwrite

For dictionaries, adding and creating new entries is the same

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

```
100
```

Dictionary Overwriting Guard

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

```python
student_records = {
    "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: Waiter

```
16_waiter.py
```

```python
1  orders = {}
2
3  order = input("Enter order: ")
4  while order:
5      count = int(input("Enter how many: "))
6
7      # Record the person's order
8      # If the order already exist, just add the count
9
10 print(orders)
```

List of Dicts

Real-life data is often more challenging to handle

Single Entry

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

```python
item = {
    'Name': 'Smartphone',
    'Info': 'Latest model smartphone',
    'Price': 70_000.00,
    'Stock': 25
}
```

Multiple Entries

By extension, you can make a list of those containers

```
1  wishlist = [
2      {
3          'Name': 'Smartphone',
4          'Info': 'Latest model smartphone',
5          'Price': 70_000.00,
6          'Stock': 25
7      },
8      {
9          'Name': 'Wireless Headphones',
10         'Info': 'Noise-canceling headphones',
11         'Price': 10_000.00,
12         'Stock': 50
13     },
14 ]
```

Multiple Entries Iteration

The first option to using a dictionary in a list of dictionaries is manual key use

```
16  for order in wishlist:
17      print("Item:")
18      print("\t Item:", order['Name'])
19      print("\t Info:", order['Info'])
20      print("\t Stock:", order['Price'])
21      print("\t Price:", order['Stock'])
22      print()
```

Multiple Entries Iteration

The second option is through a for loop

```python
16  for order in wishlist:
17      print("Item:")
18
19      for key, value in order.items():
20          print(f"\t {key}:", value)
21
22      print()
```

Quick Exercise: Wishlist (v2)

```
15_wishlist.py
```

```python
# Fill in the details of the items you plan to buy
wishlist = [
    {
        "Name": ...,
        "Info": ...,
    },
]
# Print the item details in the following format (for each item):
"""
Item:
    Name: item name
    Info: item info
    ...
"""
```

Inventory Tracker

Detailed information tracking

```python
# 17_inventory_tracker.py

def create(inventory, item):
    """Add a new item (dict) to the inventory (list[dict])"""
def read(inventory, index):
    """Return the item (dict) in the given index (int) of inventory"""
def update(inventory, index, detail_key, detail_value):
    """Change/add the key and value to the given index in inventory"""
def delete(inventory, index):
    """Remove item (dict) in the given index (int) of inventory"""
    return 0
def show(inventory):
    """Print the items and their details line-by-line"""
```

```python
# 17_inventory_tracker.py

def main():
    current_inventory = []
    command = input("Command: ")

    while command:
        # Handle command here
        # Ask the inputs here, not in the function

        # Ask for more
        command = input("Command: ")

main()
```

03

Strings

Using extra functionalities for the most used data type

Formatting

Additional formatting for f-strings

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
```

F-String Formatting

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

f"Extra text {expression}"
f"Extra text {expression :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

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

1.12   *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

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

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

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

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

```
1  number = 0.98991
2  print(f"{number:.2%}")
```

Result in Console:

98.99%

Quick Exercise: Mission Stats

18_mission_stats.py

```python
mission = "Orbiter Alpha"
distance_km = 1500000.4567
duration_days = 92.5
speed = distance_km / (duration_days * 24)
print(" Mission Log ")
print(f"Mission: {mission}")
print(f"Distance: {distance_km} km")
print(f"Duration: {duration_days} days")
print(f"Speed: {speed} km/h")

# Mission Log
# Mission: Orbiter Alpha
# Distance: 1,500,000.46 km
# Duration: 92.50 days
# Speed: 675.68 km/h
```

F-String: Text with left padding

F-strings can be used to apply layouting

f"Extra text {string :<30}"

↑

Number of characters

```
1  text = 'left aligned'
2  print(f"|{text:<30}|")
```

|left aligned |

F-String: Text with right padding

F-strings can be used to apply layouting

f"**Extra text** {string:>30}"

Number of characters

```
1  text = 'right aligned'
2  print(f"|{text:>30}|")
```

| right aligned|

F-String: Text with center padding

F-strings can be used to apply layouting

f"**Extra text** {string :^30}"

↑

Number of characters

```
1  text = 'center aligned'
2  print(f"|{text:^30}|")
```

```
|        center aligned        |
```

F-String: Text with center padding (char)

F-strings can be used to apply layouting

f"Extra text {string:=^30}"

Character for padding

```
1  text = 'center aligned'
2  print(f"|{text:=^30}|")
```

```
|========center aligned========|
```

Challenge: Mission Stats (version 2)

```
18_mission_stats.py
```

```python
mission = "Orbiter Alpha"
distance_km = 1500000.4567
duration_days = 92.5
speed = distance_km / (duration_days * 24)
print(" Mission Log ")
print(f"Mission: {mission}")
print(f"Distance: {distance_km} km")
print(f"Duration: {duration_days} days")
print(f"Speed: {speed} km/h")

#========= Mission Log =========
# Mission:        Orbiter Alpha
# Distance:     1,500,000.46 km
# Duration:          92.50 days
# Speed:            675.68 km/h
```
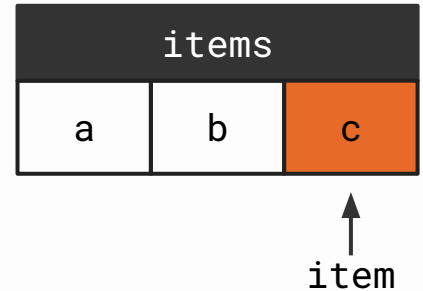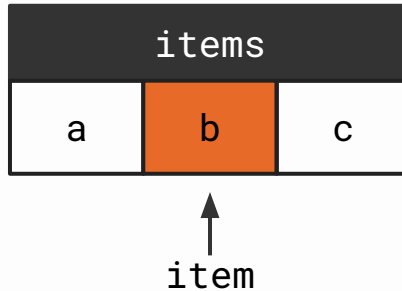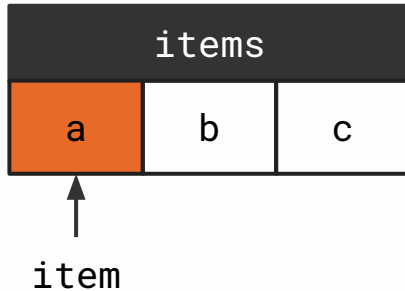
String Operations

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'
1  for item in items:
2      print(item)
```

items
a	b	c

↑
item

items
a	b	c

↑
item

items
a	b	c

↑
item

Substrings

Strings also support indexing and slicing access (not modification)

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

items										
H	e	l	l	o		W	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10

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

Quick Exercise: Special Counter

```
19_special_counter.py
```

```python
1  string = input('Enter string: ')
2  special_count = 0
3  special_char = '!@#$%^&*()'
4
5  # Add one to special_count for each special char in string
6  special_count += 1
7  print(special_count)
```

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
```

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
```

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
```

Use Case: Sanitized User Input

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

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

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

Quick Exercise: Full Agree

20_full_agree.py

```python
message = input('Agree? ')

# Print if user agrees
print('Continuing...')
```

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
```

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
```

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.isspace()
3  print(example)
4  print(all_space)
```

True

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
```

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
```

Quick Exercise: Number Check

```python
21_number_check.py
```

```python
# Ask the user for an input
user_input = input("Enter number: ")

# If user enters a valid number
user_input = int(user_input)
print(user_input + 1)

# Else
print("Please enter a valid number!")
```

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
```

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
```

Quick Exercise: Gmail Address

22_gmail_address.py

```python
# Ask the user for an input
email_input = input("Enter your email address: ")

# If valid gmail address
print("This is a valid gmail address")

# Else
print("This is NOT a valid gmail address")
```

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
```

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")
```

Quick Exercise: Number Check (v2)

21_number_check.py

```python
# Ask the user for an input
user_input = input("Enter number: ")
# Remove extra spaces

# If user enters a valid number
user_input = int(user_input)
print(user_input + 1)

# Else
print("Please enter a valid number!")
```

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
```

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
```

Quick Exercise: Number Check (v3)

21_number_check.py

```python
# Ask the user for an input
user_input = input("Enter number: ")
# Remove extra spaces
# Remove commas

# If user enters a valid number
user_input = int(user_input)
print(user_input + 1)

# Else
print("Please enter a valid number!")
```

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!']
```

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!
```

Quick Exercise: Number Check (v4)

21_number_check.py

```python
# Ask the user for an input
user_input = input("Enter number: ")
# Remove extra spaces
# Remove commas
# Remove extra spaces

# If user enters a valid number
user_input = int(user_input)
print(user_input + 1)

# Else
print("Please enter a valid number!")
```

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

```
23_crucial_dates.py
1   # You can use a custom input
2   s = "The event is on 12/15/2023, and the deadline is 01/01/2024."
3
4   # Print all of the dates mentioned
5
```

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

24_fruit_swap.py

```python
# You can use a custom input
s = "I like apple pie; pineapple is good too, apple is my favorite fruit."

# Replace every instance of "apple" with "buko"
# I like buko pie; pineapple is good too, buko is my favorite fruit.
```

Case Closed

Excellent! I cried. "Elementary," said he

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
```

Longest Word

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

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

`"The quick brown fox jumps"`	→	`"quick"`
`"I love programming in Python!"`	→	`"programming"`
`""`	→	`""`

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()** to write contents

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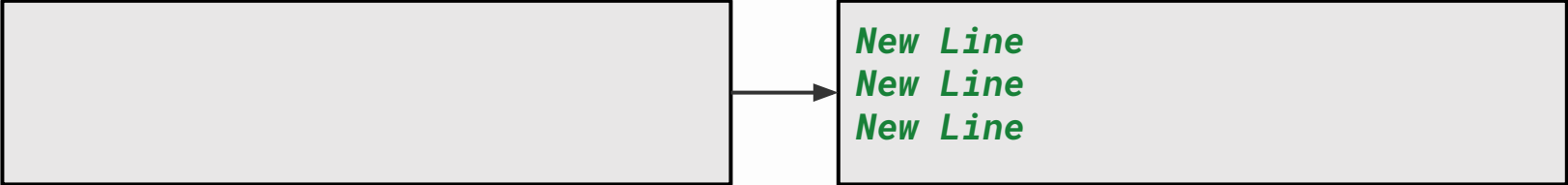
New Line

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      lines = ["New Line\n", "New Line\n", "New Line\n"]
3      file.writelines(lines)
```

New Line
New Line
New Line

```
11_cost_tracker.py

def spend(expenses, cost):
    """Add a new cost in the list of expenses"""
def refund(expenses):
    """Remove the last cost added (if any)"""
def show(expenses):
    """Print the current list of expenses line-by-line and the total"""
def save(expenses, filepath):
    """Save the expenses in the filepath"""
```

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")
```

<div>
Current Line
</div>

<div>
Current Line
New Line
</div>

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

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

Existing Line 1
Existing Line 2
Existing Line 3

file_contents

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.

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

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

→ file_contents

```python
# 11_cost_tracker.py

def spend(expenses, cost):
    """Add a new cost in the list of expenses"""
def refund(expenses):
    """Remove the last cost added (if any)"""
def show(expenses):
    """Print the current list of expenses line-by-line and the total"""
def save(expenses, filepath):
    """Save the expenses in the filepath"""
def load(expenses, filepath):
    """Return a list of expenses in the filepath"""
```

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

Unlike text handling, JSON handling requires a built-in library import

```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)

Unlike text handling, JSON handling requires a built-in library import

```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, indent=4)
```

```python
def create(inventory, item):
    """Add a new item (dict) to the inventory (list[dict])"""
def read(inventory, index):
    """Return the item (dict) in the given index (int) of inventory"""
def update(inventory, index, detail_key, detail_value):
    """Change/add the key and value to the given index in inventory"""
def delete(inventory, index):
    """Remove item (dict) in the given index (int) of inventory"""
    return 0
def show(inventory):
    """Print the items and their details line-by-line"""
def save(inventory, filepath):
    """Save the inventory (list[dict]) to a filepath"""
```

JSON Load

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

```python
import json

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

print(data)
```

```python
# 17_inventory_tracker.py

def create(inventory, item):
    """Add a new item (dict) to the inventory (list[dict])"""
def read(inventory, index):
    """Return the item (dict) in the given index (int) of inventory"""
def update(inventory, index, detail_key, detail_value):
    """Change/add the key and value to the given index in inventory"""
def delete(inventory, index):
    """Remove item (dict) in the given index (int) of inventory"""
    return 0
def show(inventory):
    """Print the items and their details line-by-line"""
def save(inventory):
    """Print the items and their details line-by-line"""
def load(inventory, filepath):
    """Return a list[dict] from a filepath"""
```

CSV Files

Handling table-like data that has rows and columns

CSV File Handling

Comma-Separated Values or CSV represent tabular data, commonly separated by commas (sometimes by other char)

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'] → 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

CSV Reading (as Lists)

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

```python
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).*

```python
import csv

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

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

Comprehensions

Syntactic Sugar for creating data structures

List Comprehension

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

```python
1  numbers = [90, 100, 20, 10, 0]
2  double_numbers = []
3  for number in numbers:
4      double_numbers.append(number * 2)
```

```python
1  double_numbers = [number * 2 for number in numbers]
```

Quick Exercise: Super Discount

```python
prices = [1_000, 10, 200, 1000, 3_000]

# Convert the numbers into half their original values
discounted_prices = []
print(discounted_prices)
```

List Comprehension (with Conditions)

```
1  tasks = {
2      'register': 'high',
3      'test': 'medium',
4      'refactor': 'low',
5  }
```

```
priority_tasks = []
for task, prio in tasks.items():
    if prio != 'low':
        priority_tasks.append(task)
```

```
priority_tasks = [task for task, prio in tasks.items() if prio != 'low']
```

Data Pipeline

Comprehensions are often used to develop pipelines or step-by-step instructions

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

Quick Exercise: Big Words

```
27_big_words.py
```

```python
1   # You can use a custom message using input()
2   sentence = "I like big data and AI models"
3
4   # Find all the words with len > 3
5   words = sentence.split()
6   big_words = []
7
8   print(big_words)
```

Clean Comprehension

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

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

def condition(number):
    return number > 10

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

Nested Data Creation

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

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

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

Formatting Control

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

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

```python
coordinates = []
for x in range(10):
    for y in range(10):
        for z in range(10):
            coordinates.append((x, y, z))
```

Quick Exercise: Standard Deck (v2)

05_standard_deck.py

```python
ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
suits = ["Hearts", "Diamonds", "Clubs", "Spades"]
"""
Create a list of possible pairing of ranks and suits
A of Hearts
2 of Hearts
3 of Hearts
...
K of Hearts
A of Diamonds
2 of Diamonds
3 of Diamonds
...
"""
```

06

Lab Session

Defining and handling data

Deck of Cards

Deck of Cards

```python
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

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

def add_bottom(deck: list[str], other: 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"""

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)"""
```

Personal Playlist

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 - Code Structure

```
1  def add(song, playlist):
2      """Add song to playlist"""
3  def remove(song, playlist):
4      """Remove song from playlist (if there)"""
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 save(playlist, filepath):
7      """Save current playlist to filepath"""
8  def load(filepath):
9      """Load a new playlist from filepath and return it"""
10 def playlist_app():
11     """
12     While user doesn't want to stop, keep asking for command
13     then do the task requested
14     """
15 playlist_app()
```

Playlist App Function

```python
def playlist_app():
    """
    While user doesn't want to stop, keep asking for command
    then do the task requested
    """

    playlist = []
    end = False

    while not end:
        user_choice = input("Select command: ")

        # Ask all inputs in the playlist_app() function to make functions simple
        if user_choice == "add":
            new_song = input("Enter song name: ")
            add(new_song, playlist)
        ...
```

Task Management

Initial Work: Starting Task

```
tasks = {
    "Alice": [
        {"title": "Fix bugs", "status": "ongoing", "priority": "critical"},
        {"title": "Write docs", "status": "todo", "priority": "medium"}
    ],
    "Bob": [
        {"title": "Update UI", "status": "done", "priority": "low"}
    ],
    "Charlie": [
        {"title": "Test code", "status": "todo", "priority": "high"}
    ]
}
```

Task Management Functions

```python
def show_all_tasks(queue):
    """Add cards in other to the last parts of deck"""
def add_task(queue, user_name, task_name, status, priority):
    """Add new task to user with given status and priority"""
def update_task_status(queue, user_name, task_name, new_status):
    """Return list of task name for user"""
def list_user_tasks(queue, user_name):
    """Return list of task name for user"""
def get_priority_summary(queue):
    """Return dict of count (how many low, medium, high, etc.)"""
def get_user_stats(queue, user_name):
    """Return dict of stats (tasks held, tasks done, etc.)"""
def add_user(queue,user_name):
    """Add new user_name from queue (if not there)"""
def remove_user(queue,user_name):
    """Remove user_name from queue (if there)"""
```

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