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

Comprehension

Iteration Shortcut

05

File Handling

Data outside code

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

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

```
A
2
3
...
10
J
Q
K
```

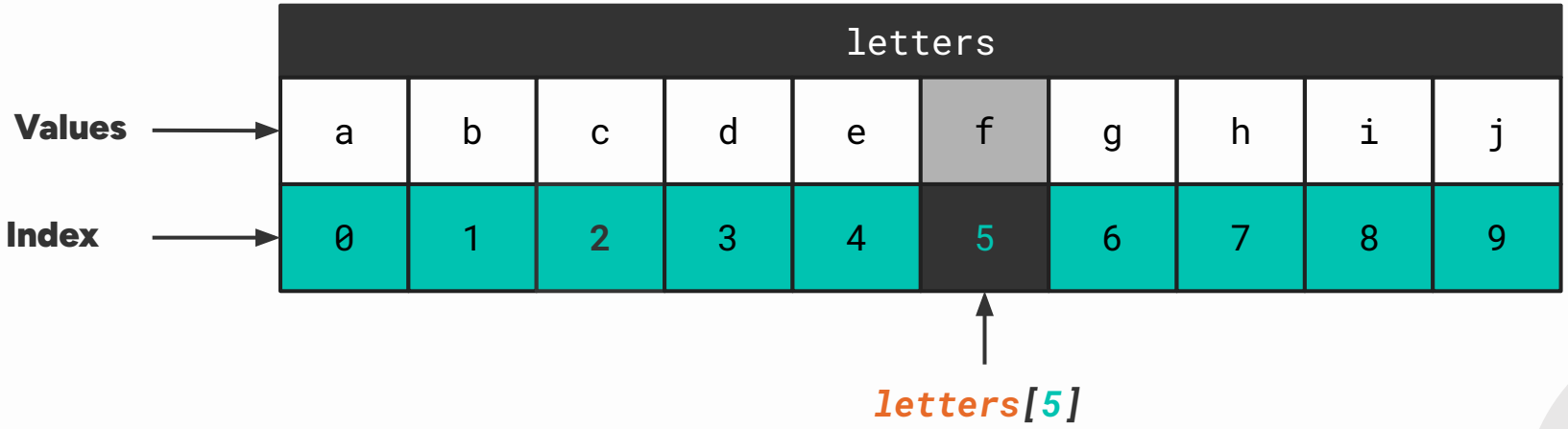
Index Logic

Always remember to start at zero

Item Access

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

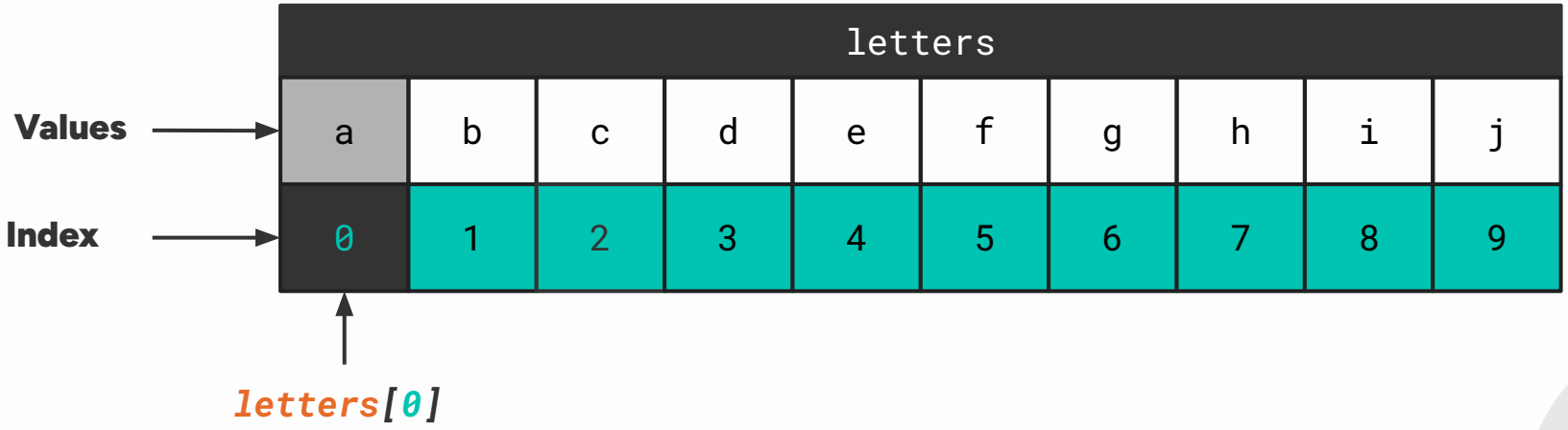
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[5])
```



Item Access

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

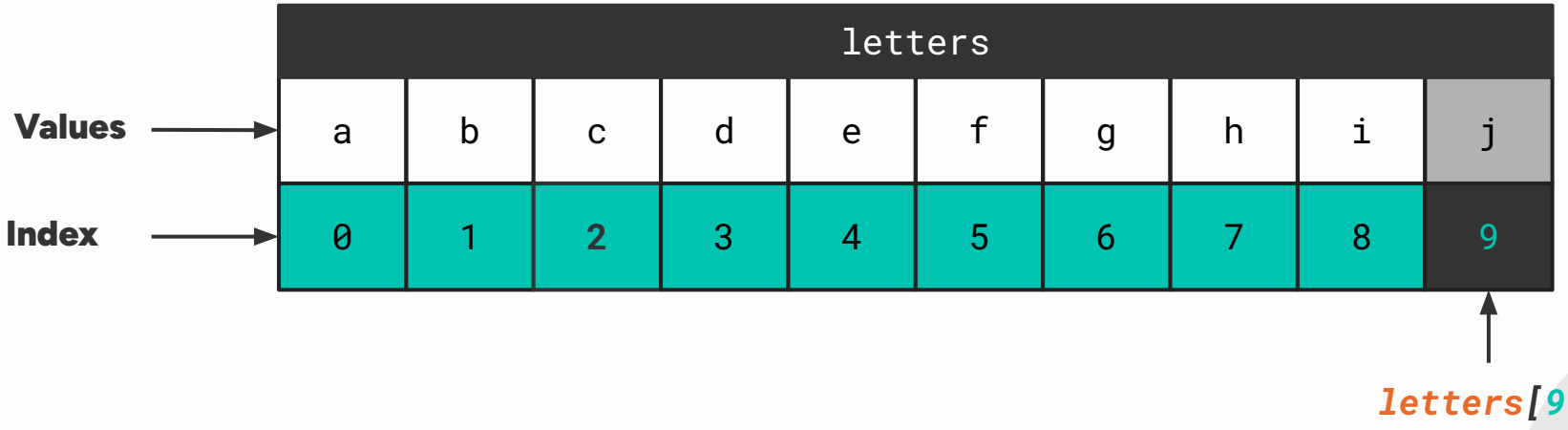
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
2 print(letters[0])
```



Item Access

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

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



Item Access

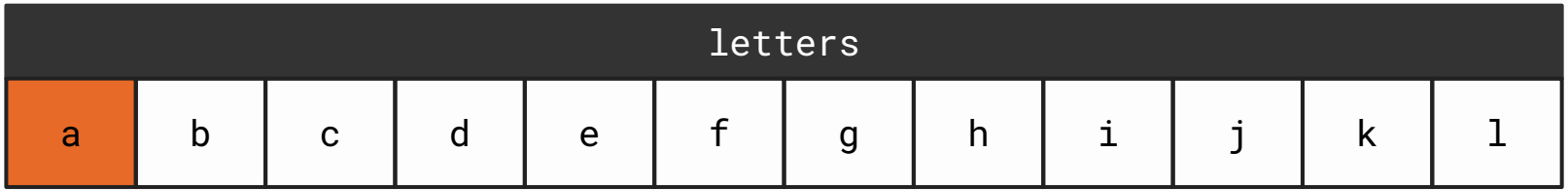
Specific values can be accessed in a list by using the list name, square brackets, and index.

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

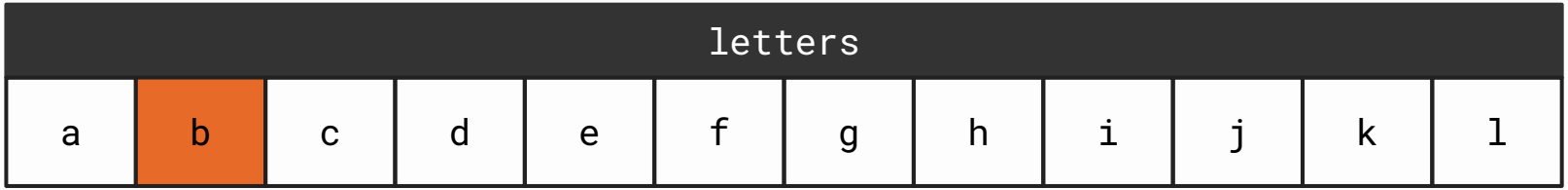
	letters									
Values →	a	b	c	d	e	f	g	h	i	j
Index (+) →	0	1	2	3	4	5	6	7	8	9
Index (-) →	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

ASAN NGA BA AKO SAYO?

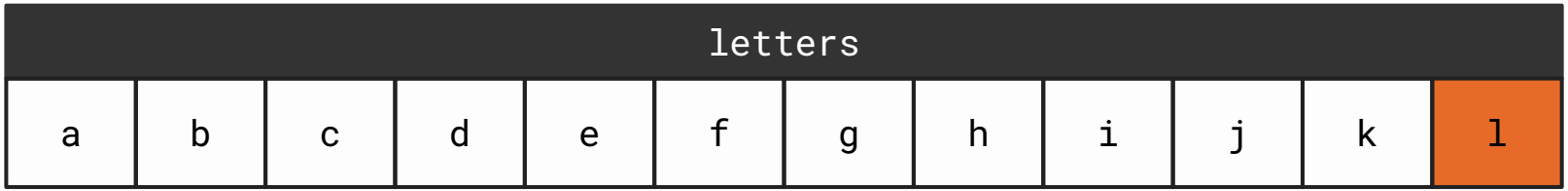
Find the Index



Find the Index



Find the Index



Find the Index

Quick Exercise: Price List

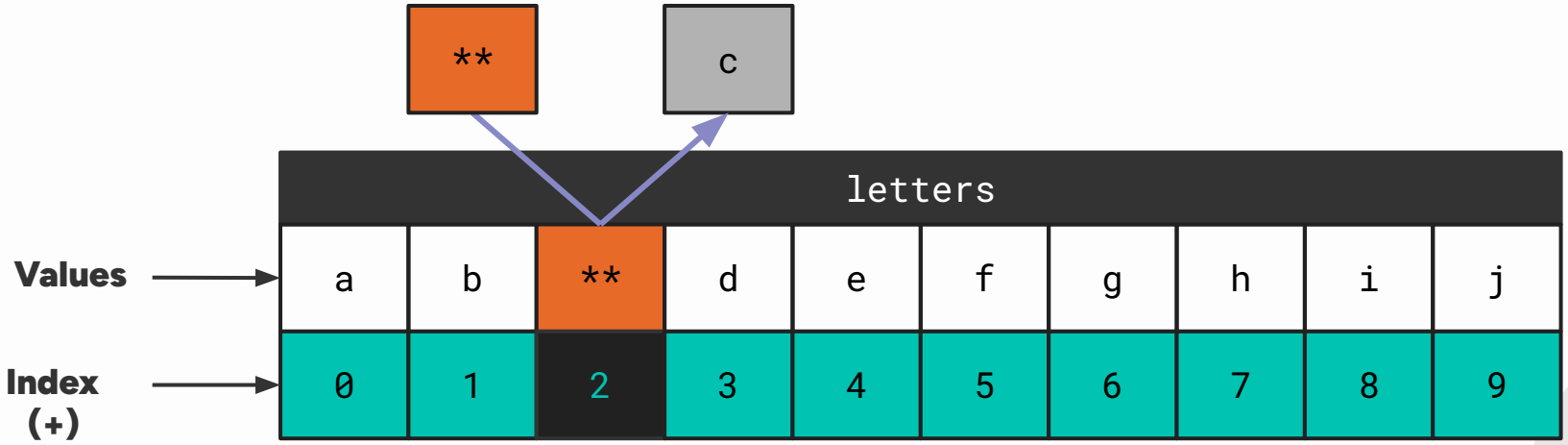
01_price_list.py

```
1  prices = [10_000, 20, 3_000, 3, -200, 1_000]
2
3  # Print the first, third, and last item
4  print("Current Price:")
5  print("Current Price:")
6  print("Current Price:")
```

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: Price List (v2)

01_price_list.py

```
1  prices = [10_000, 20, 3_000, 3, -200, 1_000]
2
3  # Print the first, third, and last item
4  print("Current Price:")
5  print("Current Price:")
6  print("Current Price:")
7
8  # Change the first, third, and last values to half the price
9
10 # Print the first, third, and last item again to see new price
11 print("New Price:")
12 print("New Price:")
13 print("New Price:")
```

Tuple Definition

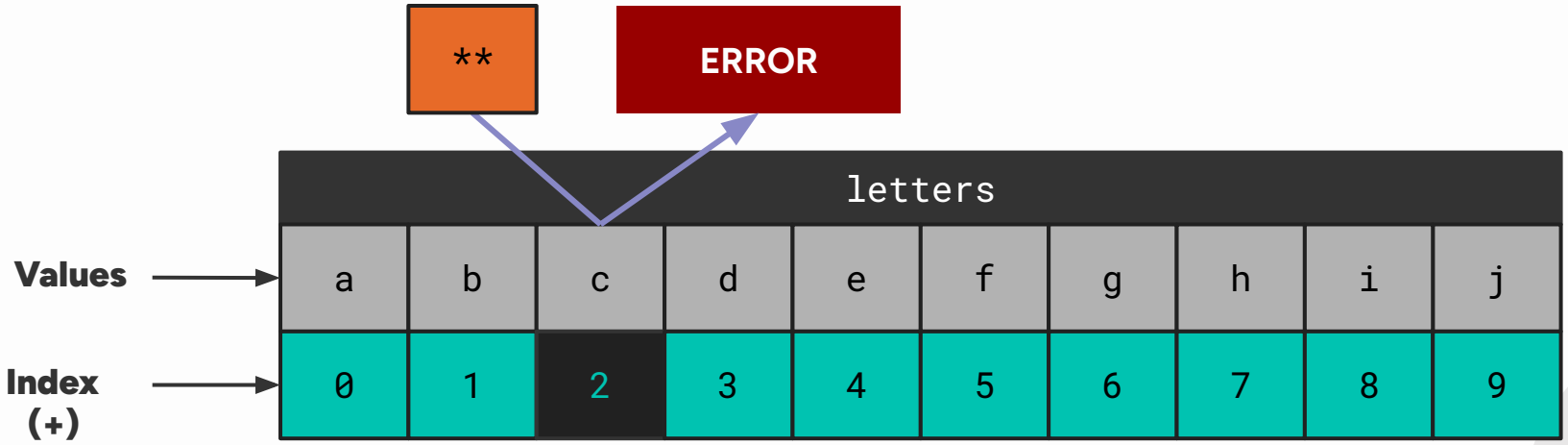
A tuple is a static, ordered collection of items, defined using parentheses and commas

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

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]
```

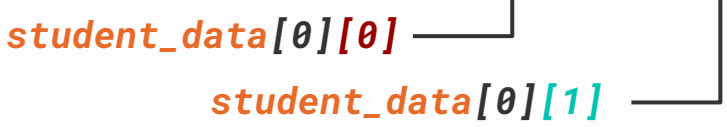
You can also directly access it by chaining indexing immediately

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

Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

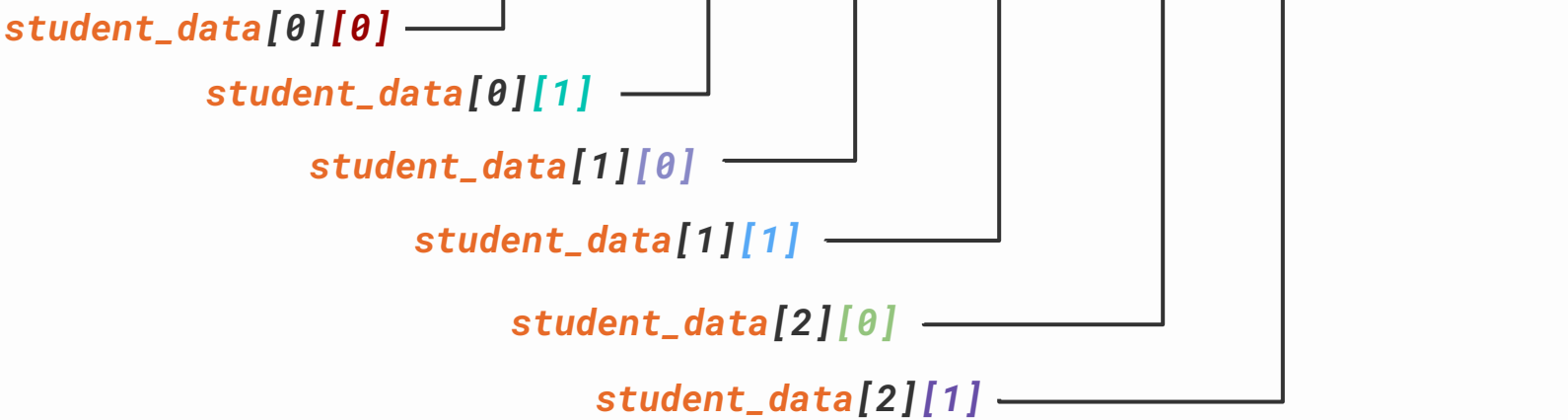
```
1  student_data = [("Maria", 98), ("Pedro", 30), ("Bax", 10)]
```



Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

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



Group Inside a Group

Lists and tuples can also contain lists or tuples inside them

```
1  student_data = [
2      ("Maria", 98),
3      ("Pedro", 30),
4      ("Bax", 10),
5  ]
6
7
```

RC	0	1
0	Maria	98
1	Pedro	30
2	Bax	10

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


NASAAN KA NA
PolyEast RECORDS
LYRIC VIDEO

Find the Index

students						
0	0	Choco	1	12	2	A
1	0	Paper	1	99	2	B
2	0	Cards	1	54	2	C
3	0	Wires	1	33	2	D

```
1  inventory = [
2      ("Choco", 12, "A"),
3      ("Paper", 99, "B"),
4      ("Cards", 54, "C"),
5      ("Wires", 33, "D"),
6  ]
```

Find the Index

students						
0	0	Choco	1	12	2	A
1	0	Paper	1	99	2	B
2	0	Cards	1	54	2	C
3	0	Wires	1	33	2	D

```
1  inventory = [
2      ("Choco", 12, "A"),
3      ("Paper", 99, "B"),
4      ("Cards", 54, "C"),
5      ("Wires", 33, "D"),
6  ]
```

Find the Index

students						
0	0	Choco	1	12	2	A
1	0	Paper	1	99	2	B
2	0	Cards	1	54	2	C
3	0	Wires	1	33	2	D

```
1  inventory = [
2      ("Choco", 12, "A"),
3      ("Paper", 99, "B"),
4      ("Cards", 54, "C"),
5      ("Wires", 33, "D"),
6  ]
```

Find the Index

students						
0	0	Choco	1	12	2	A
1	0	Paper	1	99	2	B
2	0	Cards	1	54	2	C
3	0	Wires	1	33	2	D

```
1  inventory = [
2      ("Choco", 12, "A"),
3      ("Paper", 99, "B"),
4      ("Cards", 54, "C"),
5      ("Wires", 33, "D"),
6  ]
```

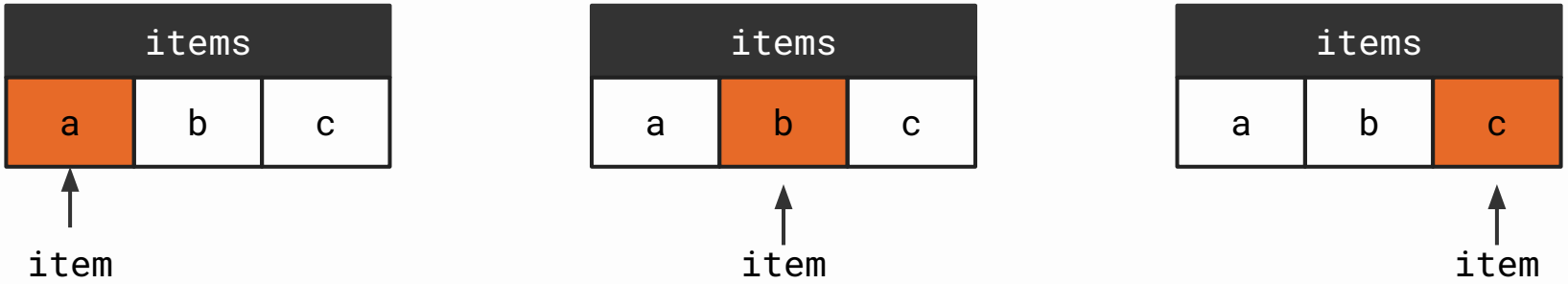
Loop Functions

Make looping more convenient

Default Looping

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

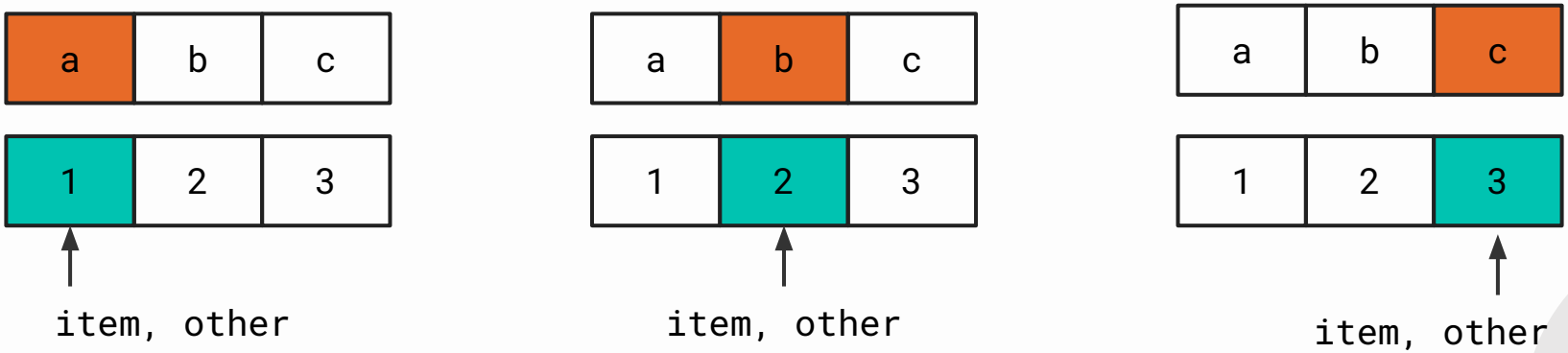
```
1  items = ('a', 'b', 'c')
2  for item in items:
3      print(item)
```



Multiple Looping

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

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



Multiple Loopings Example

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

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

```
4 for name, balance, index in zip(names, balances, indices):
5     print(f"| {index}\t| {name}\t| {balance}\tPHP\t|")
```

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

Quick Exercise: Student Records

02_student_records.py

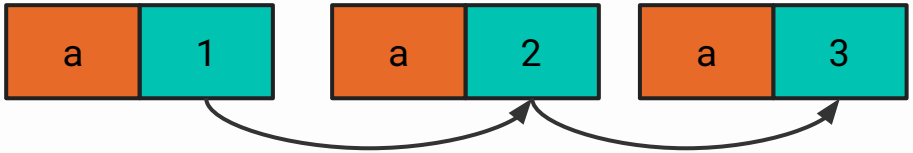
```
1  student_names = ("Juan", "Maria", "Joseph")
2  student_scores = (70, 90, 81)
3
4  """
5  Print the student scores and names in the following format
6  Student Records:
7      Student: Juan scored 70 in the exam.
8      Student: Maria scored 90 in the exam.
9      Student: Joseph scored 81 in the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

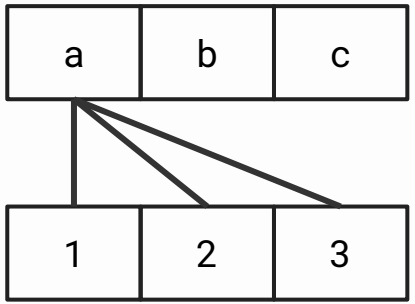
Challenge: Print the highest scorer

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)
```

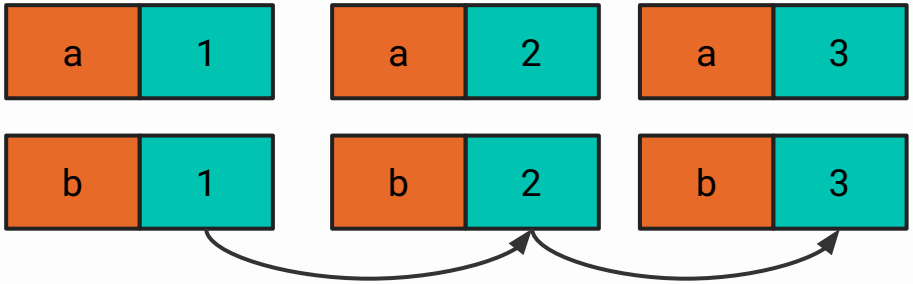


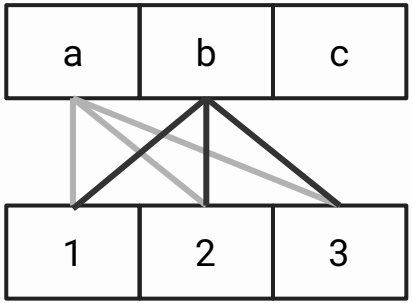


Nested Looping

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)
```

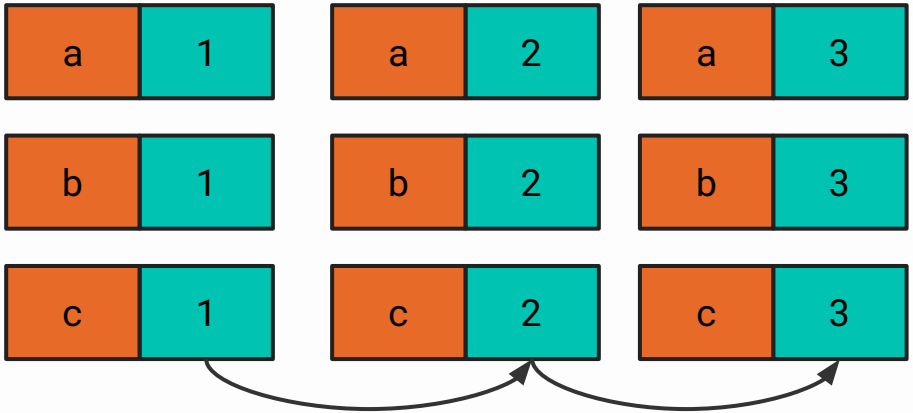


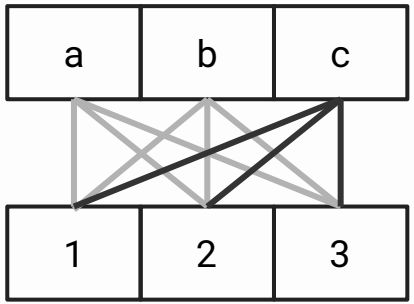


Nested Looping

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

03_standard_deck.py

```
1  ranks = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
2  suits = ("Hearts", "Diamonds", "Clubs", "Spades")
3  """
4  Print every possible pairing of ranks and suits
5  A of Hearts
6  2 of Hearts
7  3 of Hearts
8  ...
9  K of Hearts
10 A of Diamonds
11 2 of Diamonds
12 3 of Diamonds
13 ...
14 """
```

Slicing

Using index logic to take more than one element

Slicing [Start:End]

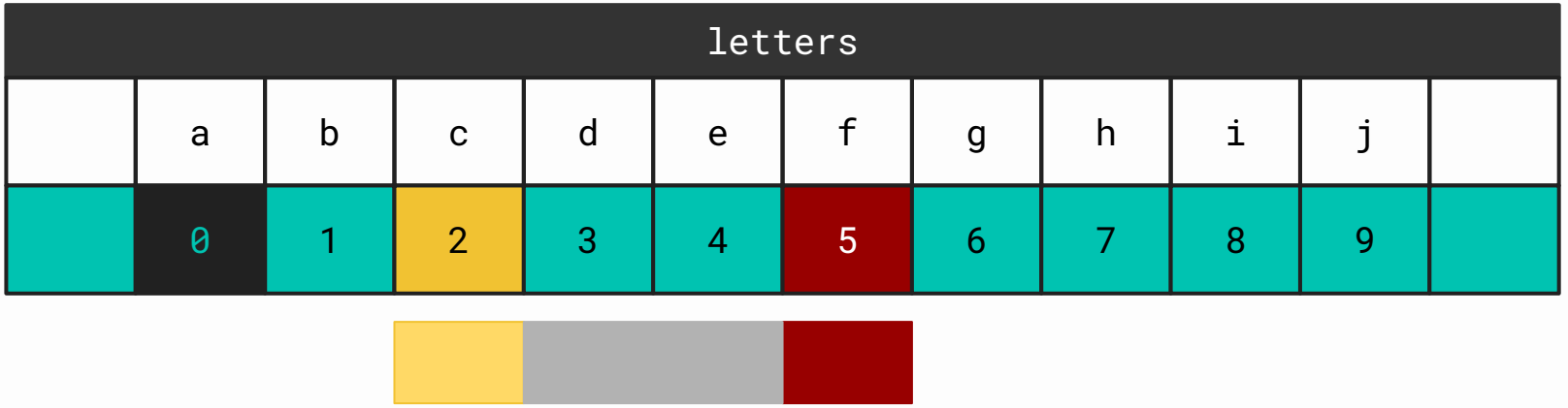
Lists and tuples can index multiple items as well using the slicing

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

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

Example 1

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



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

Example 2

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

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

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

Example 3

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

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

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

Example 4

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

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

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

Quick Exercise: Card Faces

04_card_faces.py

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

Quick Exercise: Second Draw

05_second_draw.py

```
1  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2
3  # Draw every other card
4  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

06_funny_binary.py

```
1  # Create the binary for letter 'h' as a list of 1's and 0's
2  binary_h = list(bin(ord('h')))
3  binary_h = binary_h[2:]
4
5  # Create the binary for letter 'a' as a list of 1's and 0's
6  binary_a = list(bin(ord('a')))
7  binary_a = binary_a[2:]
8
9  # Create the binary for 'hahaha'
10 binary = []
11 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

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

This is an equivalent statement

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

Quick Exercise: Banned

07_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

08_class_statistics.py

```
1  student_scores = [98, 75, 100, 86, 100, 3]
2
3  # Print the average score
4  average_score = None
5  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

```
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 data structure directly

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: Standard Deck (v2)

03_standard_deck.py

```
1  ranks = ('A','2','3','4','5','6','7','8','9','10','J','Q','K')
2  suits = ("Hearts", "Diamonds", "Clubs", "Spades")
3
4  deck = []
5  """
6  Create every possible pairing of ranks and suits
7  And add it to the empty list of cards
8  ...
9  """
10
11 print(deck)
```

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:

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

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

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

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: Price List (v3)

01_price_list.py

```
1  prices = [10_000, 20, 3_000, 3, -200, 1_000]
2
3  # Print the first, third, and last item
4  print("Current Price:")
5  print("Current Price:")
6  print("Current Price:")
7
8  # Change the first, third, and last values to half the price
9
10 # Print the first, third, and last item again to see new price
11 print("New Price:")
12 print("New Price:")
13 print("New Price:")
14 # Remove the negative price in the list
```

01

Cost Tracker

Implementing your own stack

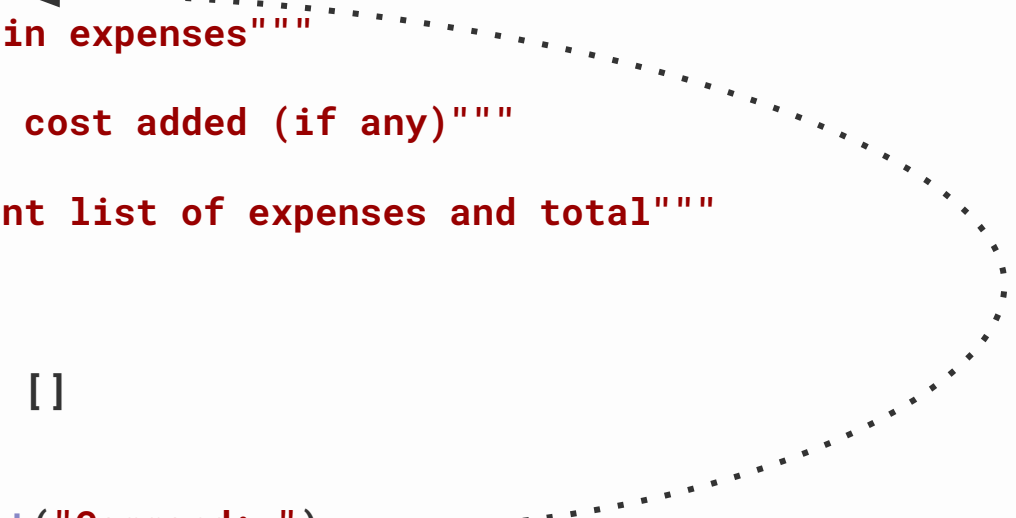
09_cost_tracker.py

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

def main():
    running = True
    current_expenses = []

    while running:
        command = input("Command: ")
        if command == "spend":
            spend(current_expenses)

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
a, b, d, c

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}
```

Example: Unique Attendance

```
1  attendee_names = set()
2
3  attendee_count = int(input("Attendee count: "))
4
5  for _ in range(attendee_count):
6      attendee_name = input("Attendee name: ")
7      attendee_names.add(attendee_name)
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}
```

Example: Unique Attendance

```
1  attendee_names = set()
2
3  attendee_count = int(input("Attendee count: "))
4
5  for _ in range(attendee_count):
6      attendee_name = input("Attendee name: ")
7      attendee_names.add(attendee_name)
8
9  exempted = input("Exempted name: ")
10 attendee_names.discard(exempted)
11
12 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
```

Example: Unique Attendance

```
1  attendee_names = set()
2
3  attendee_count = int(input("Attendee count: "))
4
5  for _ in range(attendee_count):
6      attendee_name = input("Attendee name: ")
7      attendee_names.add(attendee_name)
8
9  exempted = input("Exempted name: ")
10 attendee_names.discard(exempted)
11
12 late_attendee = attendee_names.pop()
13 print("Late: ", late_attendee)
14
15 print(attendee_names)
```

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)

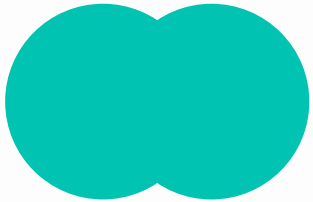
Set Operations

Borrowing from the set mathematical definition

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

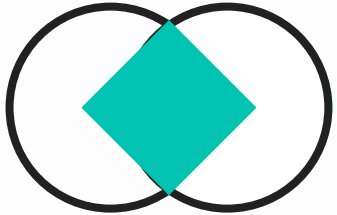


Set 2					
d	e	f	g	h	i

Set Intersection

```
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

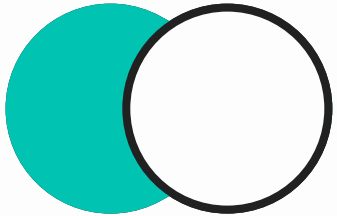


Set 2					
d	e	f	g	h	i

Set Difference

```
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

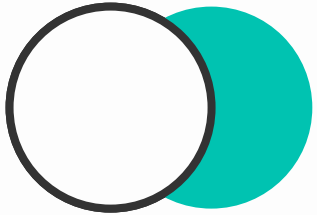


Set 2					
d	e	f	g	h	i

Set Difference (Order Matters)

```
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

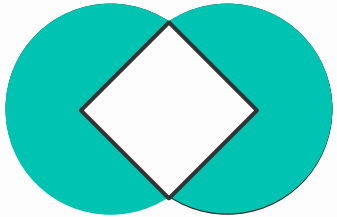


Set 2					
d	e	f	g	h	i

Set Symmetric Difference

```
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

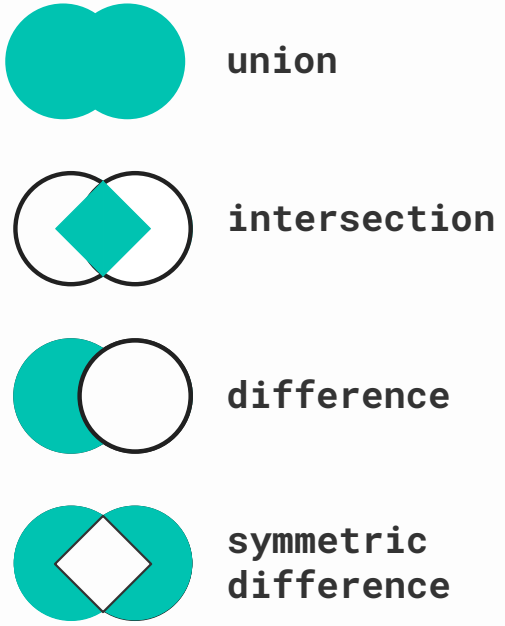


Set 2					
d	e	f	g	h	i

Quick Exercise: Gate Crashing

10_gate_crashers.py

```
1  invited = {"Ana", "Ben", "Carlo", "Dani"}
2  attended = {"Ben", "Carlo", "Ely"}
3
4  # Who are all the involved members?
5  print("Involved Members:")
6
7  # Who was absent?
8  print("Absent:")
9
10 # Who gatecrashed?
11 print("Not enrolled but attended:")
12
13 # Who was invited and attended
14 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.


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

Example 01: Form Data

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

```
1  form_data = {
2      "first_name": "Juan",
3      "last_name": "Dela Cruz",
4      "age": 25,
5      "newsletter": True
6  }
```

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

11_country_codes.py

```
1  # Add more country codes
2  country_codes = {
3      "PH": "Philippines",
4      "US": "United States",
5  }
6
7  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**

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

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

```
81
```

Quick Exercise: Country Codes (v2)

11_country_codes.py

```
1  # Add more country codes
2  country_codes = {
3      "PH": "Philippines",
4      "US": "United States",
5  }
6
7  # Print the country for the given country code
8  country_code = input("Enter country code: ")
9  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**

```
1  student_records = {
2      "Juan": 70,
3      "Maria": 98,
4      "Joseph": 81,
5      "Elise": 80
6  }
7
8  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 (v3)

11_country_codes.py

```
1  # Add more country codes
2  country_codes = {
3      "PH": "Philippines",
4      "US": "United States",
5  }
6
7  # Print the country for the given country code
8  # If the key is not found, print Unknown
9  country_code = input("Enter country code: ")
10 print(country_codes)
```

Dictionary Iteration (Keys)

The dictionary keys can be accessed using the keys method

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

Quick Exercise: Country Codes (v4)

11_country_codes.py

```
1  # Add more country codes
2  country_codes = {
3      "PH": "Philippines",
4      "US": "United States",
5  }
6
7  # Print the country for the given country code
8  # If the key is not found, print Unknown
9  country_code = input("Enter country code: ")
10 print(country_codes)
11
12 # Print all codes
```

Dictionary Iteration (Values)

The dictionary values can be accessed using the values method

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

Quick Exercise: Country Codes (v5)

11_country_codes.py

```
1  # Add more country codes
2  country_codes = {
3      "PH": "Philippines",
4      "US": "United States",
5  }
6
7  # Print the country for the given country code
8  # If the key is not found, print Unknown
9  country_code = input("Enter country code: ")
10 print(country_codes)
11
12 # Print all codes
13
14 # Print all countries
```

Dictionary Iteration (Key-Value)

Both key and values can be accessed using the items method

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

Quick Exercise: Wishlist

12_wishlist.py

```
1  # Fill in the details of the item you plan to buy
2  order = {
3      "Name": ...,
4      "Info": ...,
5  }
6
7  # Print the item details in the following format:
8  """
9  Order:
10     Name: item name
11     Info: item info
12     ...
13 """
```

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.

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

```
81
```

Quick Exercise: Waiter

13_waiter.py

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

```
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

```
wishlist = [
    {
        'Name': 'Smartphone',
        'Info': 'Latest model smartphone',
        'Price': 70_000.00,
        'Stock': 25
    },
    {
        'Name': 'Wireless Headphones',
        'Info': 'Noise-canceling headphones',
        'Price': 10_000.00,
        'Stock': 50
    },
]
```

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("Order:")
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

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

Quick Exercise: Wishlist (v2)

12_wishlist.py

```
1  # Fill in the details of the items you plan to buy
2  wishlist = [
3      {
4          "Name": ...,
5          "Info": ...,
6      },
7  ]
8  # Print the item details in the following format (for each item):
9  """
10 Item:
11     Name: item name
12     Info: item info
13     ...
   """
```

H2

Inventory Tracker

Detailed information tracking

14_inventory_tracker.py

```
def add(inventory):
    """TODO:
        Ask for item name, info, and stock
        Create a dictionary with key: name, info, stock
        Add that dictionary to inventory
    """
def remove(inventory):
    """TODO:
        Ask for item index (int)
        Remove item in that index in inventory
    """
def read(inventory):
    """TODO:
        Ask for item index (int)
        Show item in that index in inventory
    """
def show(inventory):
    """TODO: Print the items line-by-line"""
```

03

Strings

Using extra functionalities for the most used data type

Formatting

Additional formatting for f-strings

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

15_mission_stats.py

```python
mission = "Orbiter Alpha"
distance_km = 1500000.4567
days = 92.5
speed = distance_km / (days * 24)
completion = 0.35123

print(" Mission Log ")
print(f"Mission: {mission}")
print(f"Distance: {distance_km} km")
print(f"Duration: {days} days")
print(f"Speed: {speed} km/h")
print(f"Completion: {completion}")
```

```
Mission Log
Mission: Orbiter Alpha
Distance: 1,500,000.46 km
Duration: 92.50 days
Speed: 675.68 km/h
Completion: 35.123%
```

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 (v2)

15_mission_stats.py

```
1  mission = "Orbiter Alpha"
2  distance_km = 1500000.4567
3  days = 92.5
4  speed = distance_km / (days * 24)
5  completion = 0.35123
6
7  print(" Mission Log ")
8  print(f"Mission: {mission}")
9  print(f"Distance: {distance_km} km")
10 print(f"Duration: {days} days")
11 print(f"Speed: {speed} km/h")
12 print(f"Completion: {completion}")
```

```
====== Mission Log ======
Orbiter Alpha ===========
Distance: 1,500,000.5 km
Duration: 92.50 days
Speed: 675.68 km/h
Completion: 35.123%
========================
```

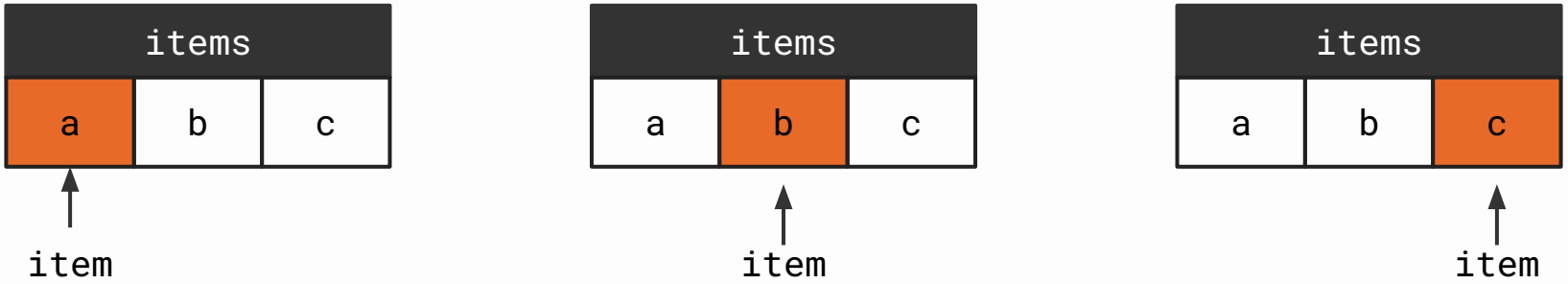
Character Handling

String formatting for one letter at a time

String Looping

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

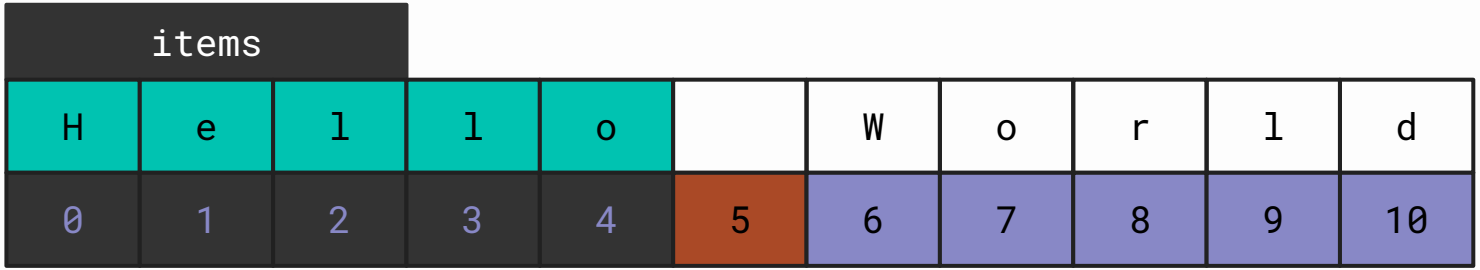
```
items = 'abc'
for item in items:
    print(item)
```



Substrings

Strings also support indexing and slicing access (not modification)

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



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

16_special_counter.py

```
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)
```

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 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 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 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
```

Use Case: Sanitized User Input

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

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

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

09_cost_tracker.py

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

def main():
    running = True
    current_expenses = []

    while running:
        command = input("Command: ")
        if command == "spend":
            spend(current_expenses)

main()
```

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

17_number_check.py

```
1  # Ask the user for an input
2  user_input = input("Enter number: ")
3
4  # If user enters a valid number
5  user_input = int(user_input)
6  print(user_input + 1)
7
8  # Else
9  print("Please enter a valid number!")
```

Word Handling

Common string methods to handle complex formatting issues

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

18_gmail_address.py

```
1  # Ask the user for an input
2  email_input = input("Enter your email address: ")
3
4  # If valid gmail address
5  print("This is a valid gmail address")
6
7  # Else
8  print("This is NOT a valid gmail address")
```

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? ")
2  clean_input = user_input.lower().strip()
3  if clean_input == "yes":
4      print("Proceeding")
```

Quick Exercise: Number Check (v2)

17_number_check.py

```
1  # Ask the user for an input
2  user_input = input("Enter number: ")
3  # Remove extra spaces
4
5  # If user enters a valid number
6  user_input = int(user_input)
7  print(user_input + 1)
8
9  # Else
10 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 = "123,456,789"
```

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

```
123,456,789
123456789
```

Quick Exercise: Number Check (v3)

17_number_check.py

```
1  # Ask the user for an input
2  user_input = input("Enter number: ")
3  # Remove extra spaces
4  # Remove commas
5
6  # If user enters a valid number
7  user_input = int(user_input)
8  print(user_input + 1)
9
10 # Else
11 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!']
```

Quick Exercise: Quick Attendance

19_quick_attendance.py

```
1  attendees = input("Enter all attendees: ")
2
3  # TODO: Print attendee names one at a time
```

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: Manual Path

20_manual_path.py

```
folders = [
    'python_bootcamp',
    'day_02',
    '03_strings',
    'manual_path.py'
]

# Create a proper filepath to the given file
# python_bootcamp/day_02/03_strings/manual_path.py
path = ''
print(path)
```

Regex

Non-linear way to handle string matching with exceptions

Regular Expressions

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

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

Regular Expressions

Here is the syntax to handle more than one special character

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

Regex Find

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

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

Quick Exercise: Crucial Dates

21_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 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.

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

Quick Exercise: Fruit Swap

22_fruit_swap.py

```
1  # You can use a custom input
2  s = "I like apple pie; pineapple is good too, apple is my favorite fruit."
3
4  # Replace every instance of "apple" with "buko"
5  # I like buko pie; pineapple is good too, buko is my favorite fruit.
```

04

Comprehensions

Syntactic Sugar for creating data structures

List Comprehension

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

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

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

storage process source

Quick Exercise: Super Discount

23_super_discount.py

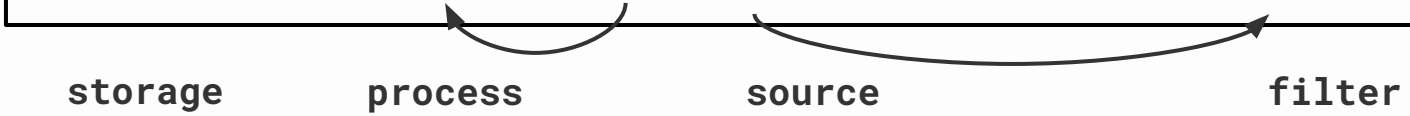
```
1  prices = [1_000, 10, 200, 1000, 3_000]
2
3  # Convert the numbers into half their original values
4  discounted_prices = []
5  print(discounted_prices)
```

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']
```



Quick Exercise: Big Words

24_big_words.py

```
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)
```

Data Pipeline

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

```
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)
```

Clean Comprehension

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

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

Nested Data Creation

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

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

```
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

```
1  ranks = ['A','2','3','4','5','6','7','8','9','10','J','Q','K']
2  suits = ["Hearts", "Diamonds", "Clubs", "Spades"]
3  """
4  Create a list of possible pairing of ranks and suits
5  A of Hearts
6  2 of Hearts
7  3 of Hearts
8  ...
9  K of Hearts
10 A of Diamonds
11 2 of Diamonds
12 3 of Diamonds
13 ...
14 """
```

05

File Handling

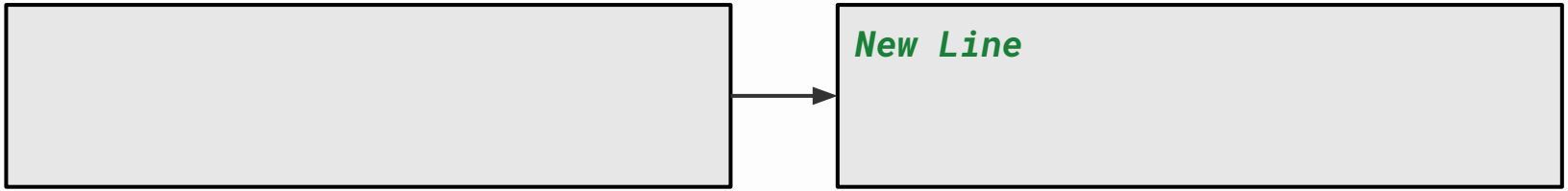
More permanent approach to data

Text Files

The most common and well-known file type

Writing Text File

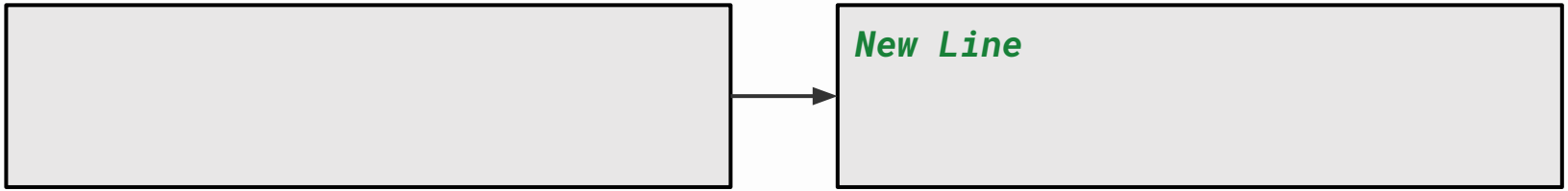
```
1 file = open("test.txt", "w")
2 file.write("New Line")
3 file.close()
```



Writing Text File

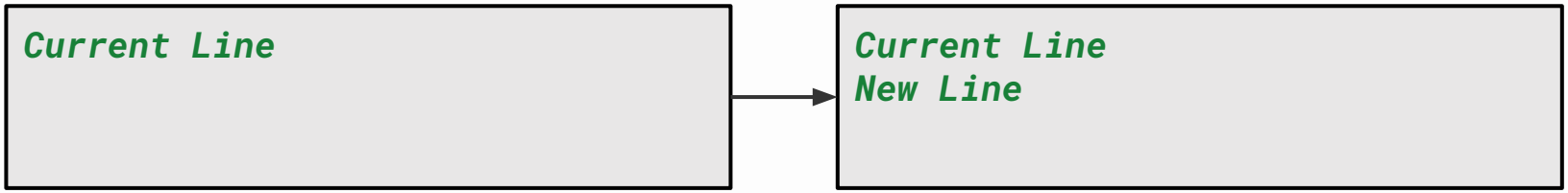
```
1 file = open("test.txt", "w")
2 file.write("New Line")
3 file.close()
```

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



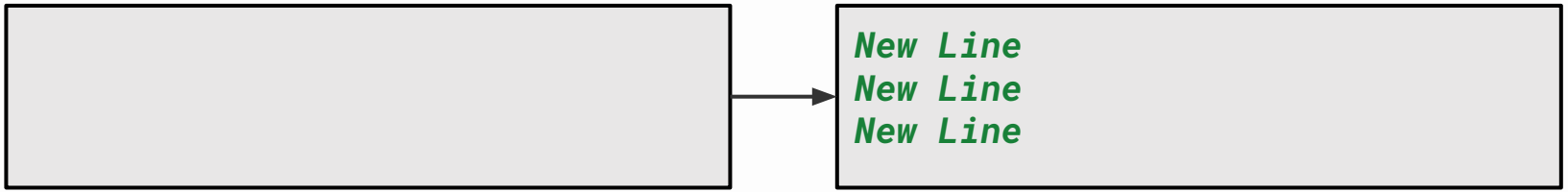
Appending Text File

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



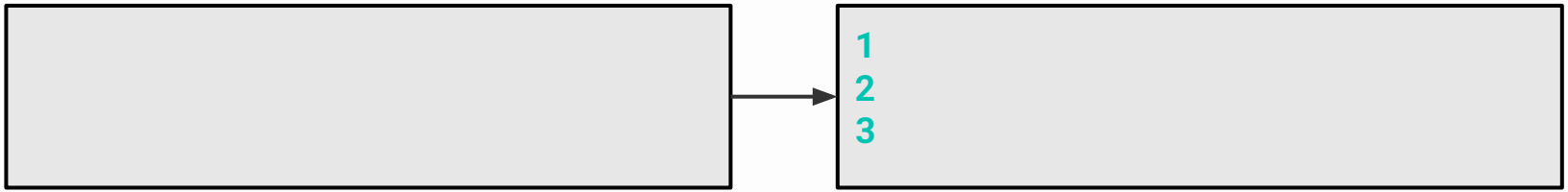
Writing Text File (Multiple String)

```
1  with open("test.txt", "w") as file:
2      lines = ["Line 1", "Line 2", "Line 3"]
3      lines = [line +"\n" for line in lines]
4      file.writelines(lines)
```



Writing Text File (Multiple Non-string)

```
1 with open("test.txt", "w") as file:
2     lines = [1, 2, 3]
3     lines = [str(line) +"\n" for line in lines]
4     file.writelines(lines)
```



09_cost_tracker.py

```
def spend(expenses):
    """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):
    """Save the expenses in a text file"""
```

Reading Text File (Full String)

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

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

```
"""Line 1
Line 2
Line 3"""
```

Reading Text File (Line by Line)

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

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

```
["Line 1\n", "Line 2\n", "Line 3\n"]
```

Reading Text File (Line by Line)

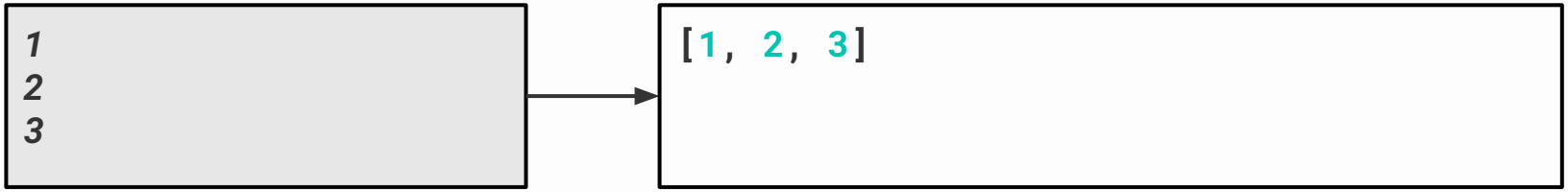
```
1 with open("test.txt", "r") as file:
2     file_contents = file.read().splitlines()
```

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

```
["Line 1", "Line 2", "Line 3"]
```

Reading Text File (Line by Line)

```
1  with open("test.txt", "r") as file:
2      numbers = [int(line) for line in file.read().splitlines()]
```



09_cost_tracker.py

```
def spend(expenses, cost):
    """Add a new cost in 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):
    """Save the expenses in a text file"""
def load():
    """Return the list of expenses from a text file"""
```

JSON

The text format of the internet

JSON File Format

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

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

JSON Dump

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

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

JSON Dump (Formatted)

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

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

14_inventory_tracker.py

```
def add(inventory):
    """..."""
def remove(inventory):
    """..."""
def read(inventory):
    """..."""
def show(inventory):
    """..."""
def save(inventory):
    """Save the inventory to a JSON file"""
```

JSON Load

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

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

14_inventory_tracker.py

```
def add(inventory):
    """..."""
def remove(inventory):
    """..."""
def read(inventory):
    """..."""
def show(inventory):
    """..."""
def save(inventory, filepath):
    """Save the inventory to a JSON file"""
def load(filepath):
    """Return the inventory from a JSON file"""
```

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)

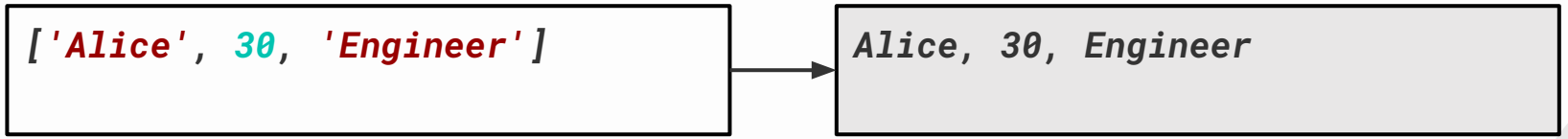
Name	Age	Occupation
Alice,	30,	Engineer
Bob,	25,	Designer
Charlie,	35,	Teacher

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)
```

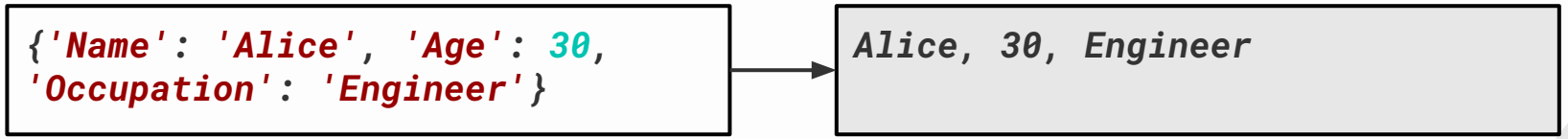


CSV Writing (with Dicts)

```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)
```



CSV Reading (as Lists)

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

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

CSV Reading (as Dicts)

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

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

06

Lab Session

Defining and handling data

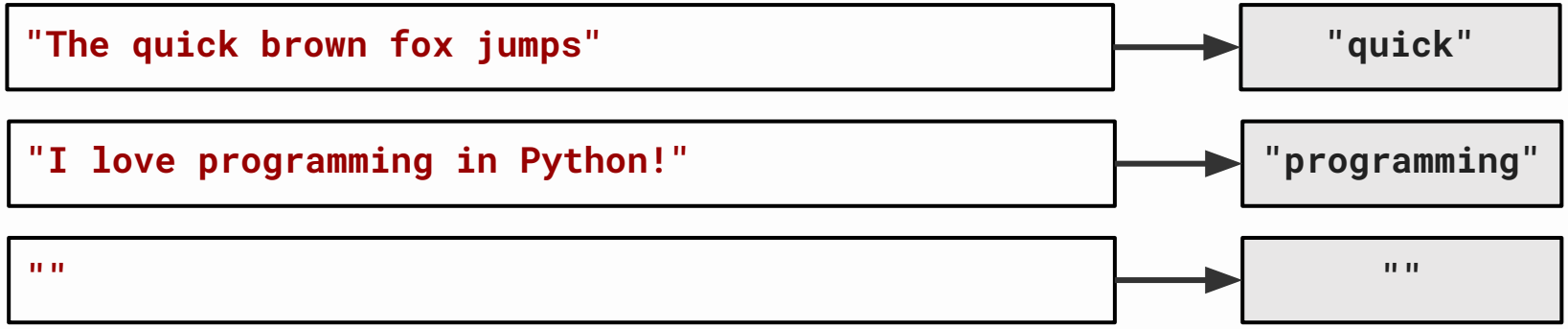
Longest Word

Pneumonoultramicroscopicsilicovolcanoconiosis

Longest Word

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

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




Deck of Cards

Deck of Cards

```
def create_deck():
    """Return a list of 52 strings containing a standard deck"""

def draw_top(deck):
    """Remove count return one card from the start from deck"""

def draw_bottom(deck):
    """Remove and return one card from the end of the deck"""

def draw_random(deck):
    """Remove and return one random cards from the deck"""

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

Challenge: Dynamic Adding

```
def add_top(deck, other):
    """Add cards in other to the first parts of deck"""

def add_bottom(deck, other):
    """Add cards in other to the last parts of deck"""

def add_random(deck, other):
    """Add cards in other randomly to deck"""

def load(filename):
    """Returns a list of cards loaded from a file"""

def save(deck, filename):
    """Saves a list of cards into a file (retrievable with load)"""
```

Public Playlist

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 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     """While loop that keeps asking for command"""
12
13 playlist_app()
```

Sneak Peak

01

Definition

Data-Centric Approach

02

Relationship

Code Reuse

03

Structure

Code Architecture

04

GUI

Introduction to Tkinter

05

Lab Session

Culminating Exercise

Python: Day 02

Data Structures