

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

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

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

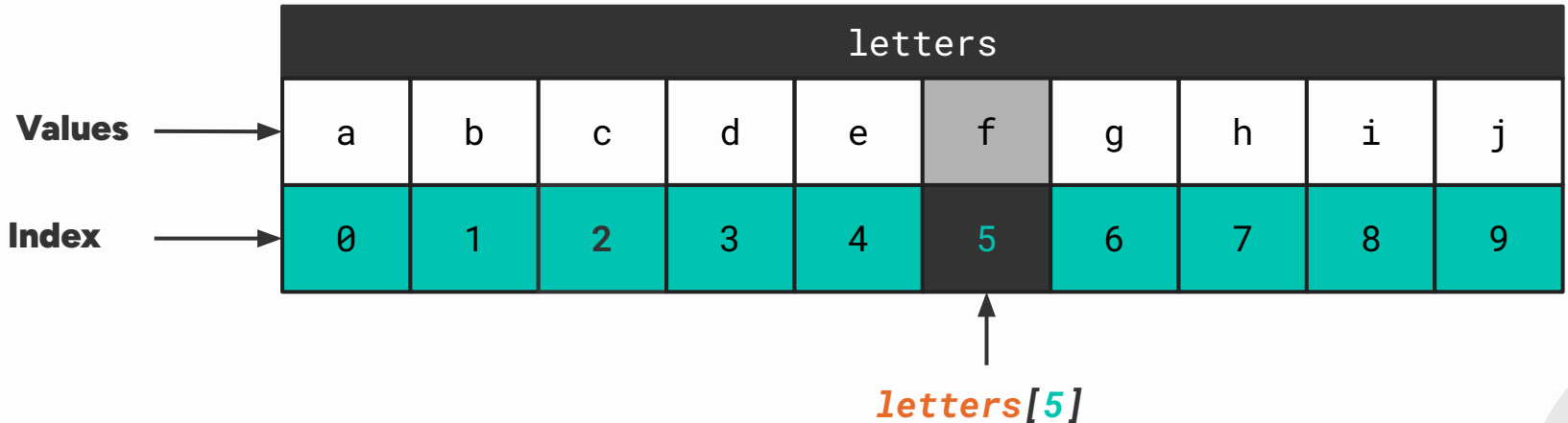
Index Logic

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

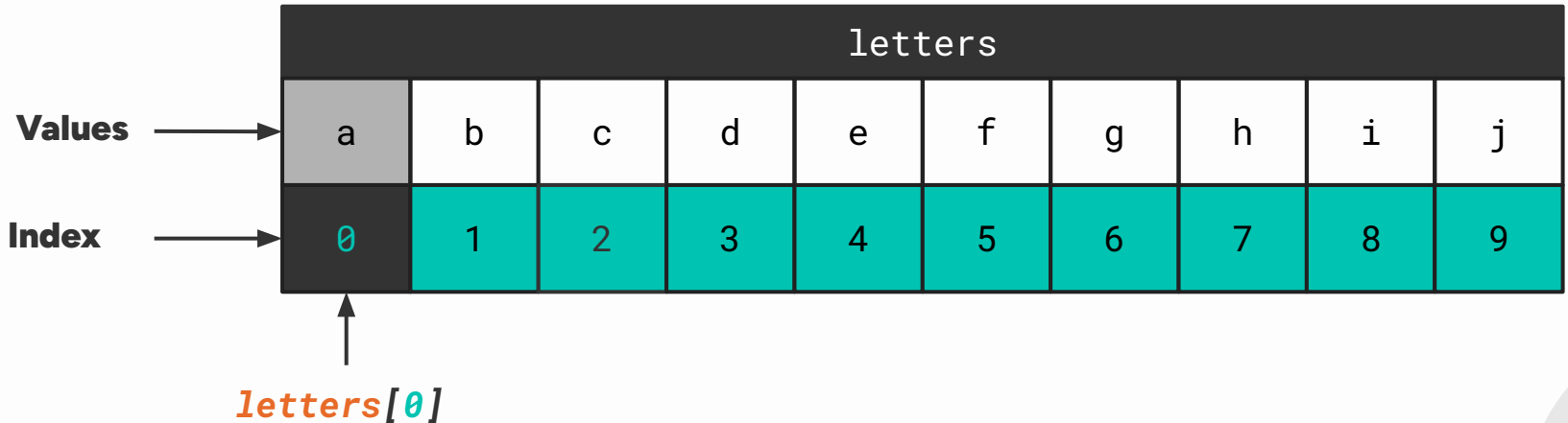
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']  
2 print(letters[5])
```



Item Access

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

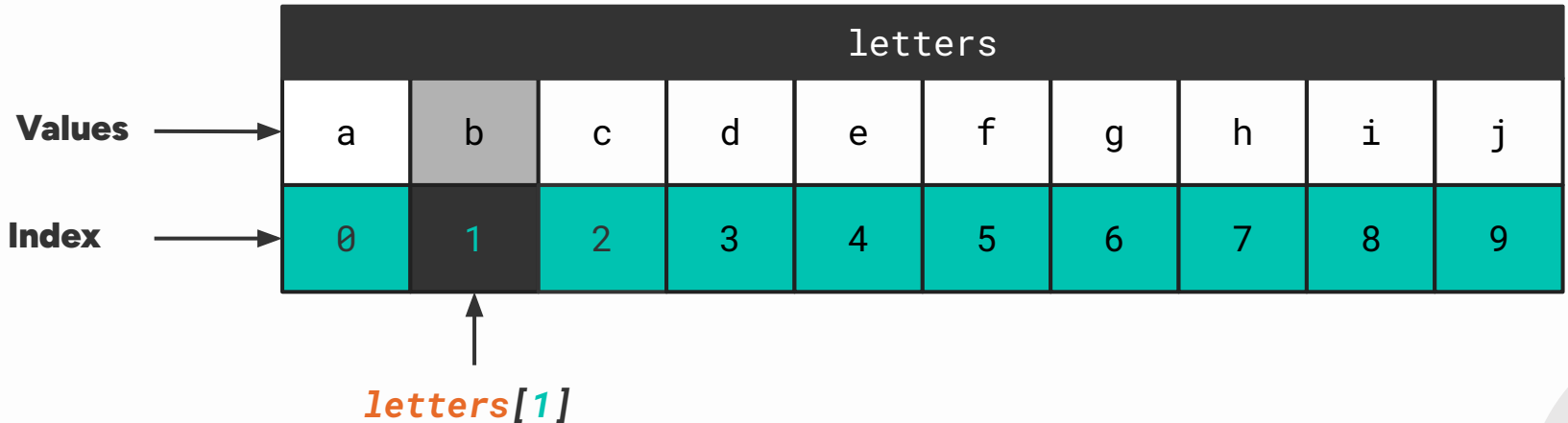
```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']  
2 print(letters[0])
```



Item Access

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

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



Item Access

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

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

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

↑
`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

Item Access

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

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

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

**ASAN NGA BA
AKO SAYO?**

**wish^{fm}
107.5**



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

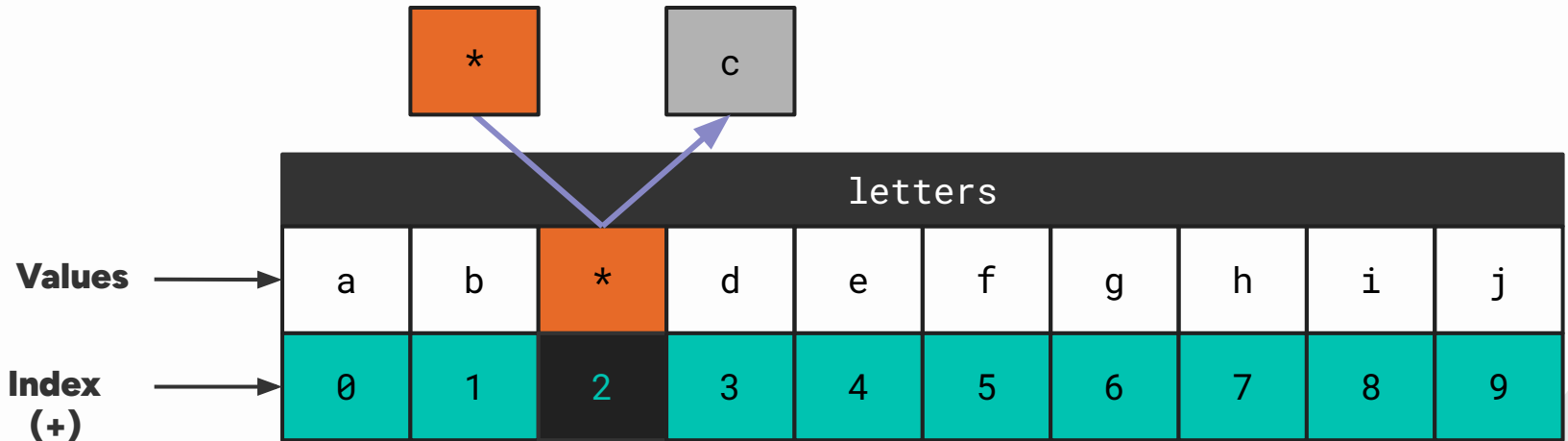
01_royal_flush.py

```
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 list  
4 print(ranks, ranks, ranks, ranks, ranks)
```

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

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 ranks = ranks
5 ranks = ranks
6 ranks = ranks
7 ranks = ranks
8 ranks = ranks
9
10 print(ranks)
```

Tuple Definition

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

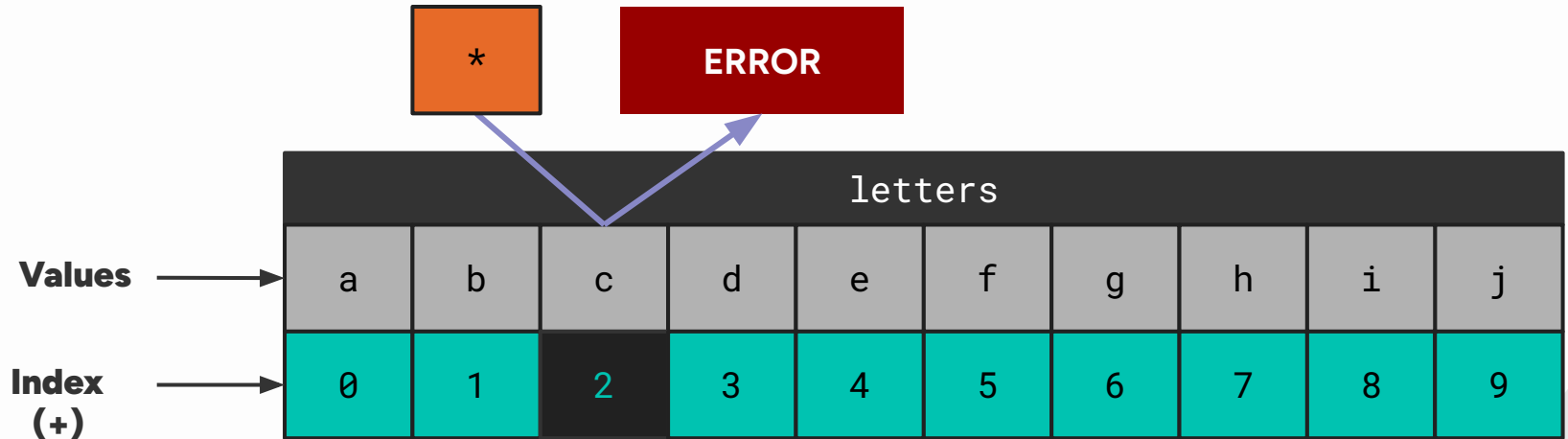
```
1 letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j')  
2 print(letters)
```

letters									
a	b	c	d	e	f	g	h	i	j

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]`

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

PolyEast
RECORDS

LYRIC VIDEO

Find the Index

students																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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																											
<table><tr><td>0</td><td>Maria</td></tr><tr><td>1</td><td>98</td></tr><tr><td>2</td><td>A</td></tr></table>	0	Maria	1	98	2	A	<table><tr><td>0</td><td>Pedro</td></tr><tr><td>1</td><td>30</td></tr><tr><td>2</td><td>B</td></tr></table>	0	Pedro	1	30	2	B	<table><tr><td>0</td><td>Bax</td></tr><tr><td>1</td><td>10</td></tr><tr><td>2</td><td>C</td></tr></table>	0	Bax	1	10	2	C	<table><tr><td>0</td><td>Theresa</td></tr><tr><td>1</td><td>61</td></tr><tr><td>2</td><td>D</td></tr></table>	0	Theresa	1	61	2	D
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]

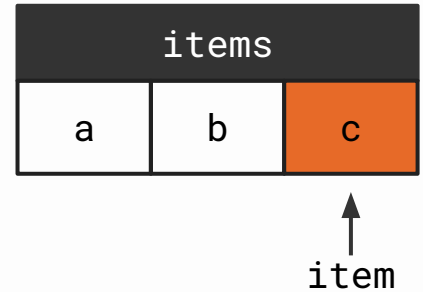
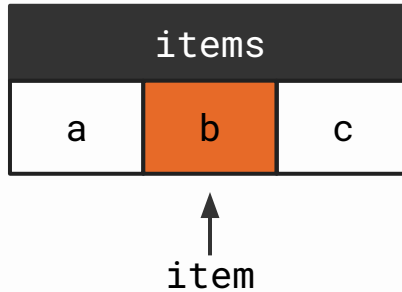
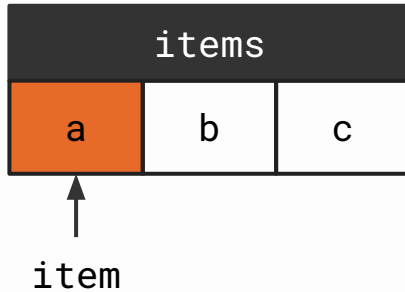
Loop Functions

Make looping more convenient

Default Looping

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

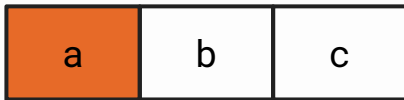
```
1 items = ('a', 'b', 'c')  
2 for item in items:  
3     print(item)
```



Multiple Looping

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

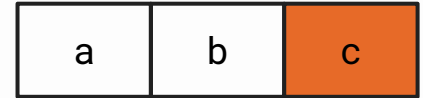
```
1 items = ('a', 'b', 'c')
2 others = (1, 2, 3)
3 for item, other in zip(items, others):
4     print(item, other)
```



item, other



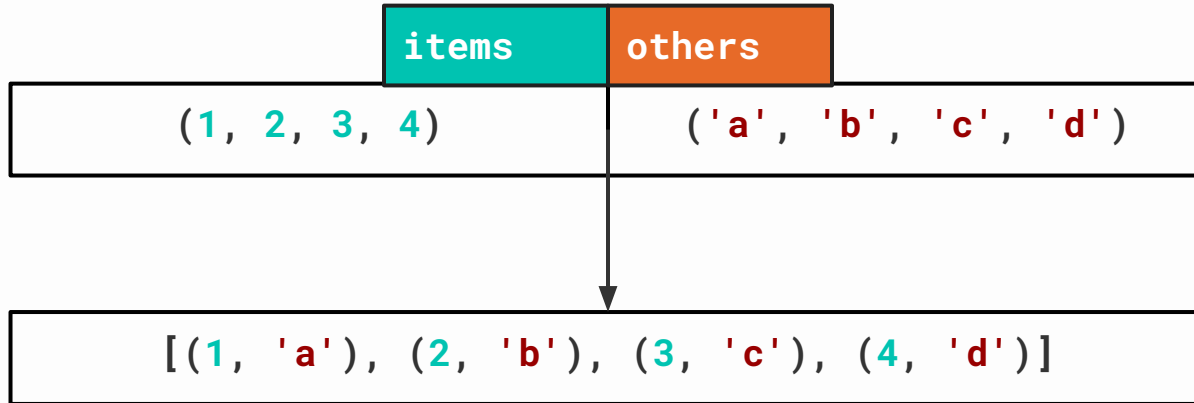
item, other



item, other

Zipping

You can group the items of multiple lists or tuples together



Multiple Loopings Example

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

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

```
4 for name, balance, id in zip(names, balances, ids):
5     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

```
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)

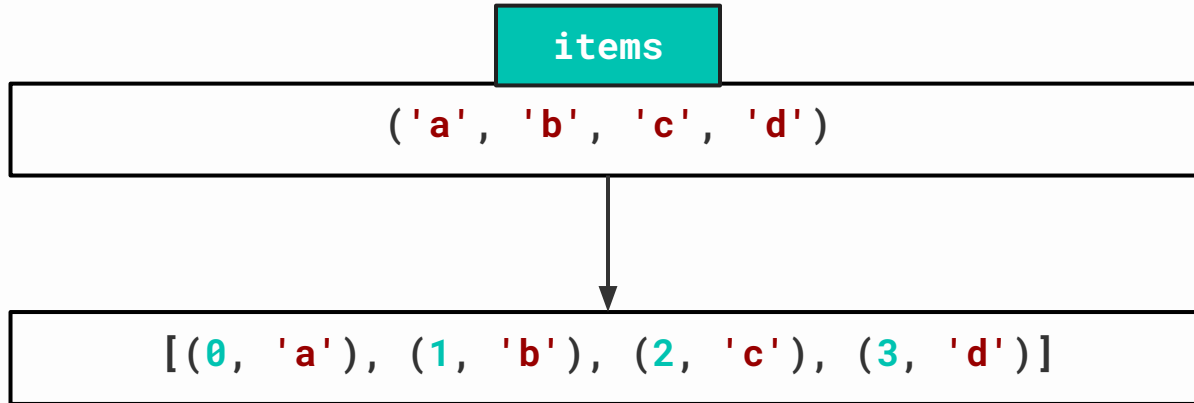
You can set the start of the enumerate function using the start parameter.

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

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

Enumerate

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



Quick Exercise: Student Records (version 2)

03_student_records.py

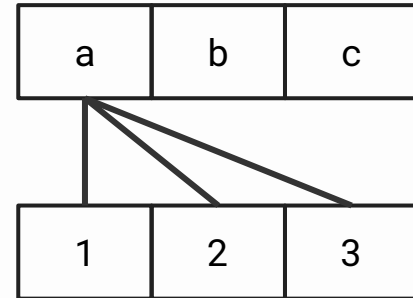
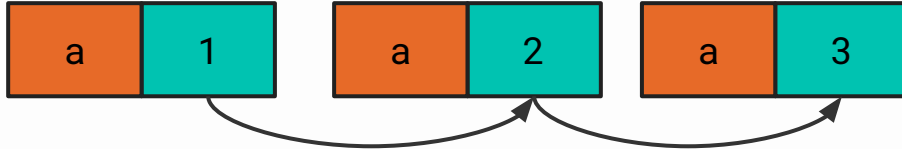
```
1 student_names = ("Juan", "Maria", "Joseph")
2 student_scores = (70, 90, 81)
3
4 """
5 Print the student scores and names in the following format
6 Student Records:
7     Student 1: Juan scored 70 in the exam.
8     Student 2: Maria scored 90 in the exam.
9     Student 3: Joseph scored 81 in the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

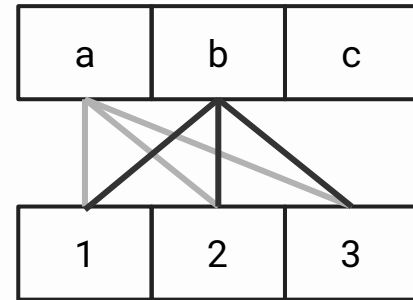
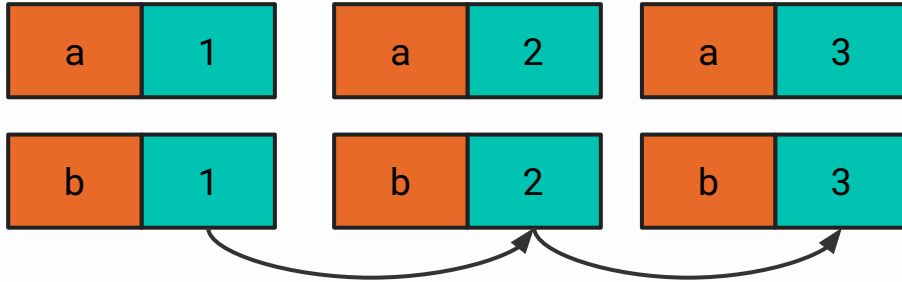
```
1 items = ('a', 'b', 'c')
2 others = (1, 2, 3)
3 for item in items:
4     for other in others:
5         print(item, other)
```



Nested Looping

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

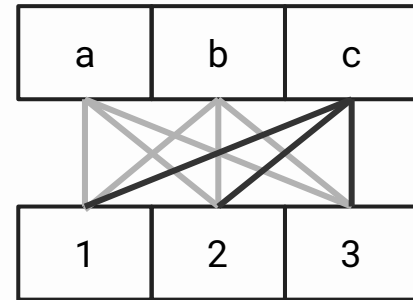
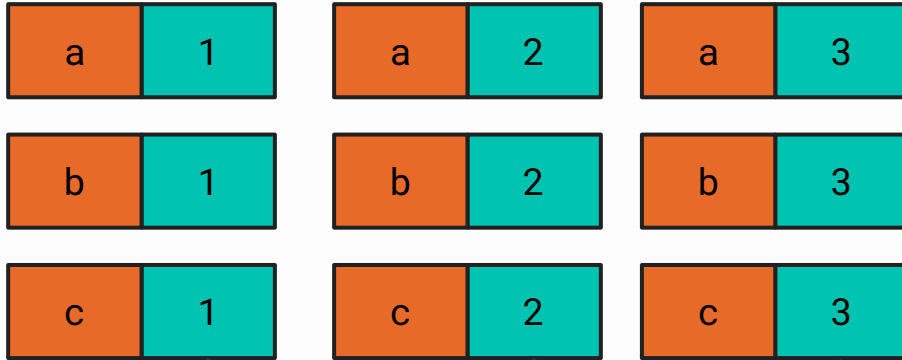
```
1 items = ('a', 'b', 'c')
2 others = (1, 2, 3)
3 for item in items:
4     for other in others:
5         print(item, other)
```



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)
```



Example: Multiplication Table

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

```
1 for i in range(1, 4):  
2     for j in range(1, 4):  
3         print(f"{i} x {j} = {i * j}")  
4  
5     # blank line between tables  
6     print()
```

```
1 x 1 = 1  
1 x 2 = 2  
1 x 3 = 3  
  
2 x 1 = 2  
2 x 2 = 4  
2 x 3 = 6  
  
3 x 1 = 3  
3 x 2 = 6  
3 x 3 = 9
```


Quick Exercise: Standard Deck

04_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]
```

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)

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' the from list  
4 print(ranks, ranks)
```


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

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_priority.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]
```

Quick Exercise: Class Statistics

08_class_statistics.py

```
1 student_scores = [98, 75, 100, 86, 100, 3]
2
3 # Print the lowest score
4 lowest_score = None
5 print(lowest_score)
6
7 # Print the highest score
8 highest_score = None
9 print(highest_score)
```


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 (version 2)

08_class_statistics.py

```
1 student_scores = [98, 75, 100, 86, 100, 3]
2
3 # Print the lowest score
4 lowest_score = None
5 print(lowest_score)
6
7 # Print the highest score
8 highest_score = None
9 print(highest_score)
10
11 # Print the average score
12 average_score = None
13 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 (version 3)

08_class_statistics.py

```
1  student_scores = [98, 75, 100, 86, 100, 3]
2
3  # Print the lowest score
4  lowest_score = None
5  print(lowest_score)
6
7  # Print the highest score
8  highest_score = None
9  print(highest_score)
10
11 # Print the average score
12 average_score = None
13 print(average_score)
14
15 # Print the rankings, highest to lowest
16 print()
```

Quick Exercise: Student Rankings

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

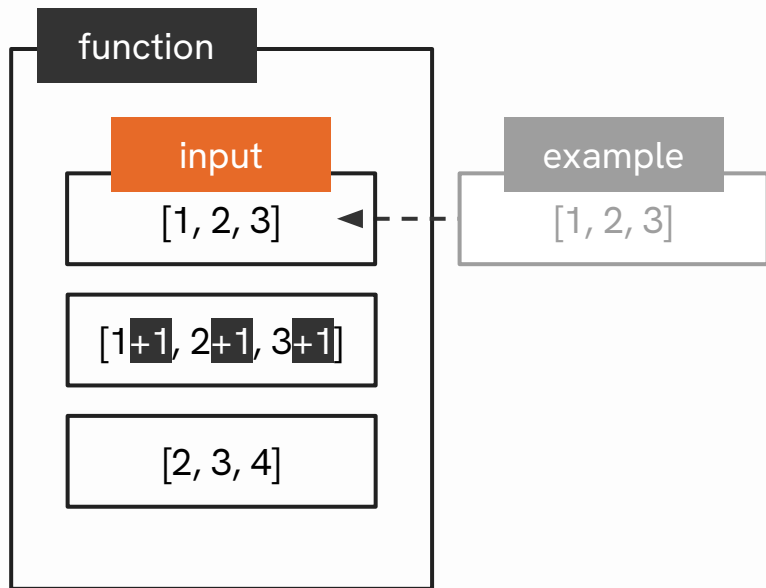
```
Student Score 1: Highest score  
Student Score 2: Second highest score  
Student Score 3: Third highest score  
...
```

Methods

Modifying the function directly

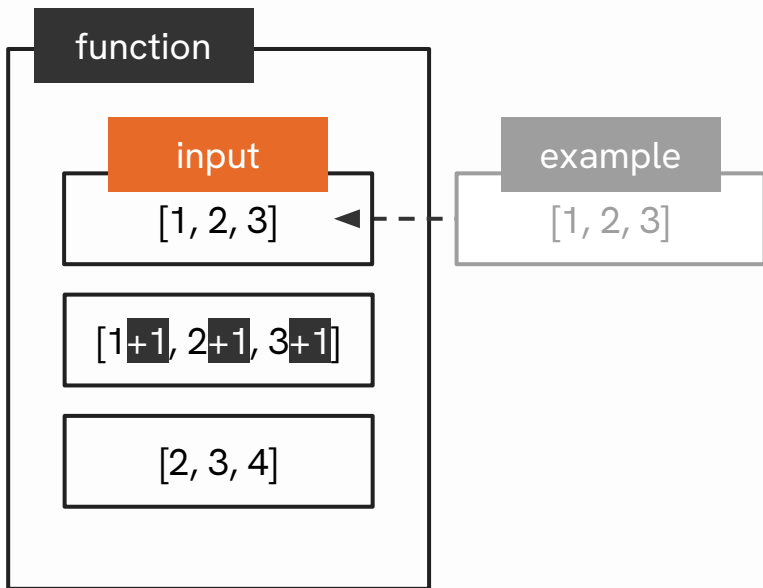
Recall: Functions

value = *function*(*value*)

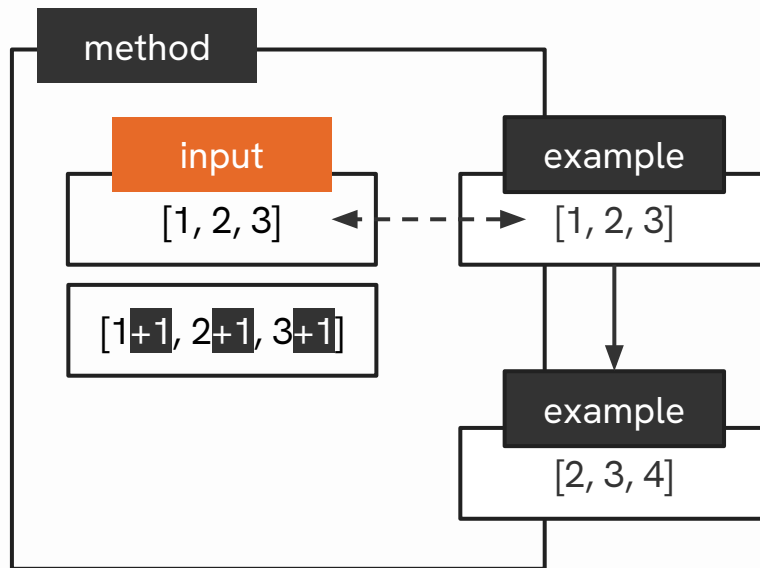


Functions vs Methods

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



```
value.method()
```



Methods Affects Readable Data


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

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

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

Best Practice: Use Function Inputs

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



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

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

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]
```

Extend Method

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

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

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

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

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

09_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)
```


Index Method

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

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

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

```
3
```

Count Method

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

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

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

```
2
```

Quick Exercise: Attendance (version 2)

09_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 # What order did you appear in the attendees?
12 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:

```
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]
```

Quick Exercise: Attendance (version 3)

09_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 # What order did you appear in the attendees?
12 print(attendee_names)
13
14 # Remove your name from the attendees
15 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 (version 4)

09_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 # What order did you appear in the attendees?
12 print(attendee_names)
13
14 # Remove your name from the attendees
15 print(attendee_names)
16
17 # Remove and print the late attendee (last attendee)
```

01

TODO List

The hello world of CRUD applications

TODO List: Setup

10_todo.py

```
tasks = []

def create(tasks: list[str], task: str):
    """Add a new task at the end of the tasks"""

def read(tasks: list[str], index: int) -> str:
    """Return the task in the index given"""

def update(tasks: list[str], index: int, new_task: str):
    """Change the value in the index to the new task"""

def delete(tasks: list[str], index: int):
    """Remove the task in the given index"""
```

Test the TODO List

10_todo.py

```
create(tasks, "Buy milk")
create(tasks, "Do homework")
create(tasks, "Sleep")

assert "Buy milk" in tasks
assert read(tasks, 1) == "Do homework"

update(tasks, 0, "Buy coffee")
assert "Buy milk" not in tasks
assert "Buy coffee" in tasks

delete(tasks, 2)
assert "Sleep" not in tasks
assert len(tasks) == 2
```

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

11_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 `discard` method 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)

11_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 # Remove your name from attendees (if there)
10
11 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)

11_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 # Remove your name from attendees (if there)
10
11 print(attendee_names)
12
13 # Pick a random raffle winner (attendee)
```

Applicable Functions

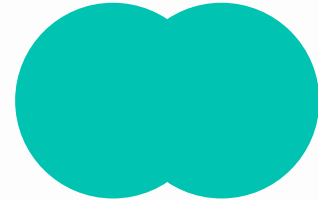
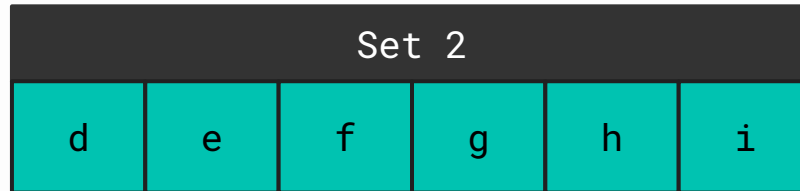
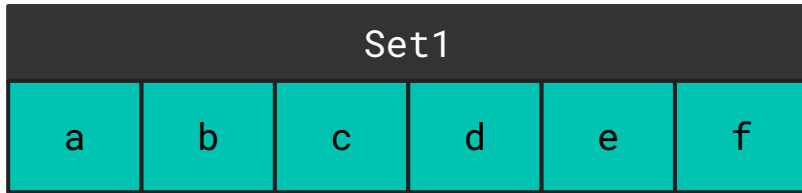
Function Usage	Behavior
<code>len(example)</code>	Returns the number of items in a set
<code>min(example)</code>	Returns the lowest value in the set. Raises <code>ValueError()</code> if empty
<code>max(example)</code>	Returns the highest value in the set. Raises <code>ValueError()</code> if empty
<code>sum(example)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

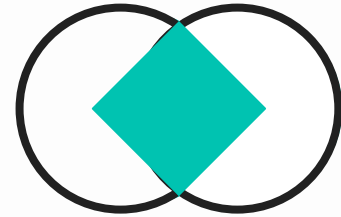


Set Intersection

```
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

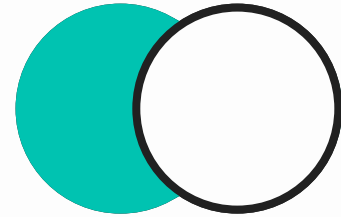


Set Difference

```
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

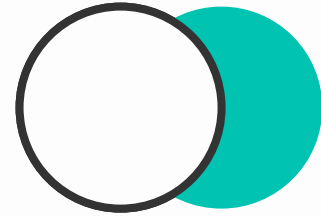


Set Difference (Order Matters)

```
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

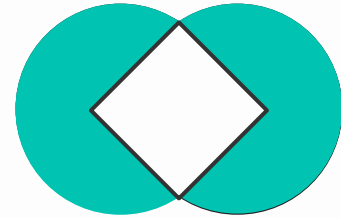


Set Symmetric Difference

```
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

12_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.



```
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

```
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

```
1 status_codes = {  
2     200: "OK",  
3     404: "Not Found",  
4     500: "Server Error"  
5 }
```

Quick Exercise: Country Codes

13_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**

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

81

Quick Exercise: Country Codes (version 2)

13_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 (version 3)

13_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 Loops

Handling a set and list at once

Dictionary Iteration (Keys)

The dictionary keys can be accessed using the `keys` method

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

Dictionary Iteration (Keys)

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

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

Quick Exercise: Country Codes (version 4)

13_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 (version 5)

13_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

14_wishlist.py

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

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 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["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

Dictionary Common Methods

Here are some of the notable methods for dictionaries

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

Quick Exercise: Time Tracker

15_time_tracker.py

```
1 tracker = {
2     "Sabrina": 7400,
3     "Rumi": 7200,
4 }
5
6 for _ in range(10):
7     runner_name = input("Enter name: ")
8     runner_time = input("Enter time: ")
9
10    # Update tracker time for runner in tracker dict
11    # Challenge: only update if new time is less than past time
12
13 print(tracker)
```

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

```
1 item = {  
2     'Name': 'Smartphone',  
3     'Info': 'Latest model smartphone',  
4     'Price': 70_000.00,  
5     'Stock': 25  
6 }
```

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:", item['Name'])
19     print("\t Info:", item['Info'])
20     print("\t Stock:", item['Price'])
21     print("\t Price:", item['Stock'])
22     print()
23
```

Multiple Entries Iteration

The second option is through a for loop

```
16 for order in wishlist:
17     print("Item:")
18
19     for key in order:
20         print("\t {key}:", item[key])
21
22     print()
```

Multiple Entries Iteration

The second option is through a for loop

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

Quick Exercise: Wishlist (version 2)

14_wishlist.py

```
1  # Fill in the details of the items you plan to buy
2  item = [
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     ...
14 """
```


H2

Cart System

Handle more than one type of information at a time

Cart System

```
1  cart = []
2
3  def add(cart, name, price, quantity):
4      """Add a new item with the following details"""
5
6  def remove(cart, index):
7      """Remove entry with key name from cart"""
8
9  def show_all(cart):
10     """Print all contents in cart"""
11
12 def show_total(cart):
13     """Calculate and print total of cart"""
```

Cart System

```
12 print("Adding items...")
13 add(cart, "Taho", 10, 3)
14 add(cart, "Isaw", 10, 5)
15 add(cart, "Kwek-kwek", 8, 4)
16 show_all(cart)
17
18 print("Removing item at index 1 (Isaw)...")
19 remove(cart, 1)
20 show_all(cart)
21
22 print("Showing total...")
23 show_all(cart)
```

03

Strings

Using extra functionalities for the most used data type

Formatting

Additional formatting for f-strings

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

98.99%

Quick Exercise: Mission Stats

15_mission_stats.py

```
1 mission = "Orbiter Alpha"
2 distance_km = 1500000.4567
3 duration_days = 92.5
4 speed = distance_km / (duration_days * 24)
5 print(" Mission Log ")
6 print(f"Mission: {mission}")
7 print(f"Distance: {distance_km} km")
8 print(f"Duration: {duration_days} days")
9 print(f"Speed: {speed} km/h")
10
11 # Mission Log
12 # Mission: Orbiter Alpha
13 # Distance: 1,500,000.46 km
14 # Duration: 92.50 days
15 # 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)

15_mission_stats.py

```
1 mission = "Orbiter Alpha"
2 distance_km = 1500000.4567
3 duration_days = 92.5
4 speed = distance_km / (duration_days * 24)
5 print(" Mission Log ")
6 print(f"Mission: {mission}")
7 print(f"Distance: {distance_km} km")
8 print(f"Duration: {duration_days} days")
9 print(f"Speed: {speed} km/h")
10
11 #===== Mission Log =====
12 # Mission:      Orbiter Alpha
13 # Distance:      1,500,000.46 km
14 # Duration:      92.50 days
15 # Speed:         675.68 km/h
```

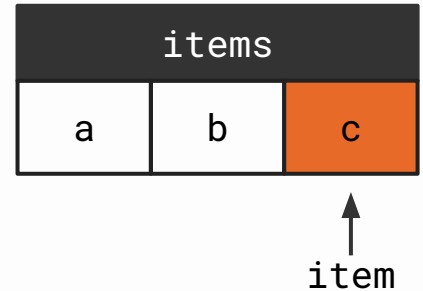
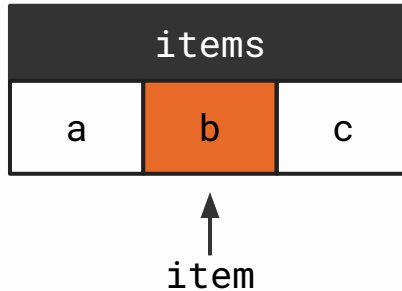
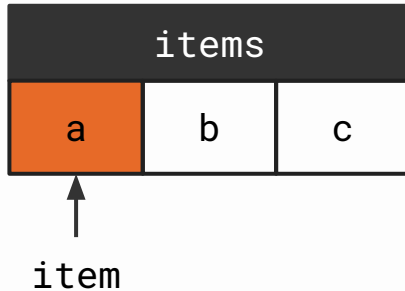

String Operations

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

```
1 items = 'abc'  
2 for item in items:  
    print(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

16_special_counter.py

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

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

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



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

Quick Exercise: Angry

17_angery.py

```
1 message = input('Enter message: ')\n2\n3 # Convert every character in message to upper case\n4 angry_message = message\n5\n6 print(angry_message)
```

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

18_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!")
```

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

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")
```

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 (version 2)

18_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 = "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 (version 3)

18_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!']
```

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 (version 4)

18_number_check.py

```
1  # Ask the user for an input
2  user_input = input("Enter number: ")
3  # Remove extra spaces
4  # Remove commas
5  # Remove extra spaces
6
7  # If user enters a valid number
8  user_input = int(user_input)
9  print(user_input + 1)
10
11 # Else
12 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
.	<code>r"c.t"</code>	Matches any single character except a newline.
*	<code>r"a*bc"</code>	Matches zero or more of the preceding character
+	<code>r"a+bc"</code>	Matches one or more of the preceding character
?	<code>r"colou?r"</code>	Matches zero or one of the preceding character
[]	<code>r"[cb]at"</code>	Matches one of the characters in square bracket
{n,m}	<code>r"a{n,m}"</code>	Matches preceding character from <code>n</code> to <code>m</code> 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

19_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.

```
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

20_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.
```

H4A

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

H4B

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
```

"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

Quick Exercise: Write Guestlist

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

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

Appending Text File

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

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

Current Line



Current Line
New Line

Quick Exercise: Append Guestlist

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

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

Reading Text File (Full String)

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

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

Existing Line 1
Existing Line 2
Existing Line 3

`file_contents`

Reading Text File (Line by Line)

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

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

Existing Line 1
Existing Line 2
Existing Line 3

file_contents

Quick Exercise: Read Guestlist

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

Attendees:

Mia Anderson

Ethan Roberts

Liam Johnson

Sophia Martinez

Olivia Davis

Noah Thompson

Alex Freze

JSON

The text format of the internet

JSON File Format

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

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

JSON Dump

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)
```


Quick Exercise: Wishlist (version 3)

14_wishlist.py

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

Quick Exercise: Wishlist (version 4)

14_wishlist.py

```
1  # Fill in the details of the items you plan to buy
2  item = [{...}, {...}, ...]
3
4  # Print the item details in the following format (for each item):
5  """
6  Item:
7      Name: item name
8      Info: item info
9      ...
10 """
11 # Save the file to a JSON file
12 # Load JSON file again into loaded_item
13 loaded_item = None
14 # Print the loaded contents from the 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)

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

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



`Alice, 30, Engineer`

CSV Writing (with Dicts)

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

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


CSV Reading (as Dicts)

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

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

Quick Exercise: Wishlist Database (Copy)

14_wishlist_database.py

```
1  # Fill in the details of the items you plan to buy
2  item = [{...}, {...}, ...]
3
4  # Save the file to a JSON file
5
6  # Load CSV file again into loaded_item
7  loaded_item = None
8
9  # Print the loaded contents from the CSV file
```

05

Comprehensions

Syntactic Sugar for creating data structures

List Comprehension

The most efficient way to generate a list of items

List Comprehension

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

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

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

Quick Exercise: Squared

22_squared.py

```
1 n = int(input("How many to generate? "))
2
3 # Generate the following list based on number_count
4 # 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, ..., n*n
5 squares = []
6 print(squares)
```

List Comprehension (with Conditions)

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

```
1 example_list = []  
2 for num in range(11):  
3     if num % 2 == 0:  
4         example_list.append(num)  
5
```

```
1 example_list = [num for num in range(11) if num % 2 == 0]
```

Example: Basic Soup

```
1 import re
2
3 html = """
4 <h1>Welcome to My Blog</h1>
5 <p>This is the first paragraph.</p>
6 <p>Contact me at info@example.com</p>
7 """
8 lines = html.splitlines()
9 lines = [re.sub(r'<.*?>', '', line) for line in lines]
10 lines = [line.strip() for line in lines if line.strip()]
11
12 print(lines)
```


Quick Exercise: Big Words

23_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)  
]  
print(coordinates)
```

```
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

```
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))
```

Example: Uno Cards

```
colors = ["Red", "Green", "Blue", "Yellow"]  
values = ["1", "2", "3", "4", "5", "6", "7", "8", "9", "R", "S", "+2"]
```

```
cards = [f"{v}-{n}" for c in colors for v in values]  
cards.extend(["Color", "Color", "Color", "Color"])  
cards.extend(["+4", "+4", "+4", "+4"])
```

```
cards = []  
for c in colors:  
    for v in values:  
        cards.append(f"{v}-{n}")  
  
cards.extend(["Color", "Color", "Color", "Color"])  
cards.extend(["+4", "+4", "+4", "+4"])
```

Quick Exercise: Standard Deck (version 2)

04_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 """
```

06

Lab Session

Defining and handling data














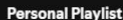
Personal Playlist

Stephen • 142 songs, 10 hr 37 min



Q Custom order

#	Title	Album	Date added	🕒
1	 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 Sunagawa, IC Horroo	Paruparo	2 weeks ago	4:57



Pwede Ba

Lola Amour

Related music videos



Madali

Lola Amour, Al...



Please Don't...

Lola Amour

About the artist

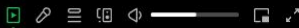


Pwede Ba
Lola Amour



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):
6     """Print the first song in the playlist (if any) and remove"""
7 def show_all(playlist):
8     """Print all contents in the playlist"""
9 def save(playlist, filepath):
10     """Save current playlist to filepath"""
11 def load(filepath):
12     """Load a new playlist from filepath and return it"""
13 def playlist_app():
14     """
15     While user doesn't want to stop, keep asking for command
16     then do the task requested
17     """
18     playlist_app()
```

Playlist App Function

```
def playlist_app():  
    """  
    While user doesn't want to stop, keep asking for command  
    then do the task requested  
    """  
  
    playlist = []  
    exit = False  
  
    while not exit:  
        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(song, playlist)  
        ...
```



Deck of Cards

Deck of Cards

```
def create_deck() -> list[str]:  
    """Return a list of 52 strings containing a standard deck"""  
  
def draw_top(deck: list[str], count: int=1) -> list[str]:  
    """Remove count return count cards from the start from deck"""  
  
def draw_bottom(deck: list[str], count: int=1) -> list[str]:  
    """Remove and return count cards from the end of the deck"""  
  
def draw_random(deck: list[str], count: int=1) -> list[str]:  
    """Remove and return count random cards from the deck"""  
  
def show(deck):  
    """Print all cards in deck"""
```

Challenge: Dynamic Adding

```
def add_top(deck: list[str], other: list[str]):  
    """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)"""
```

W O R D
F I N D

Initial Work: Word Bank

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

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


Game Setting: Word Selection

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

Current Categories:

1. Fruits
2. Animals
3. Countries

Choose a category: `user input`

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

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

Actual Game: Letter Guessing

Show the selected word as underscores

Ask the user for a letter input.

Enter letter: **user input**

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

pp

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

Additions: Quality of Life Updates

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

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

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

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

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

Challenge: Dynamic Gameplay

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

Current Categories:

1. Fruits
2. Animals

Options:

1. Add category
2. Add word
3. Start Game

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

You guessed the word: apple

You made 0 incorrect guesses. That's amazing!

Do you want to play again (y/n)? `user_input`

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