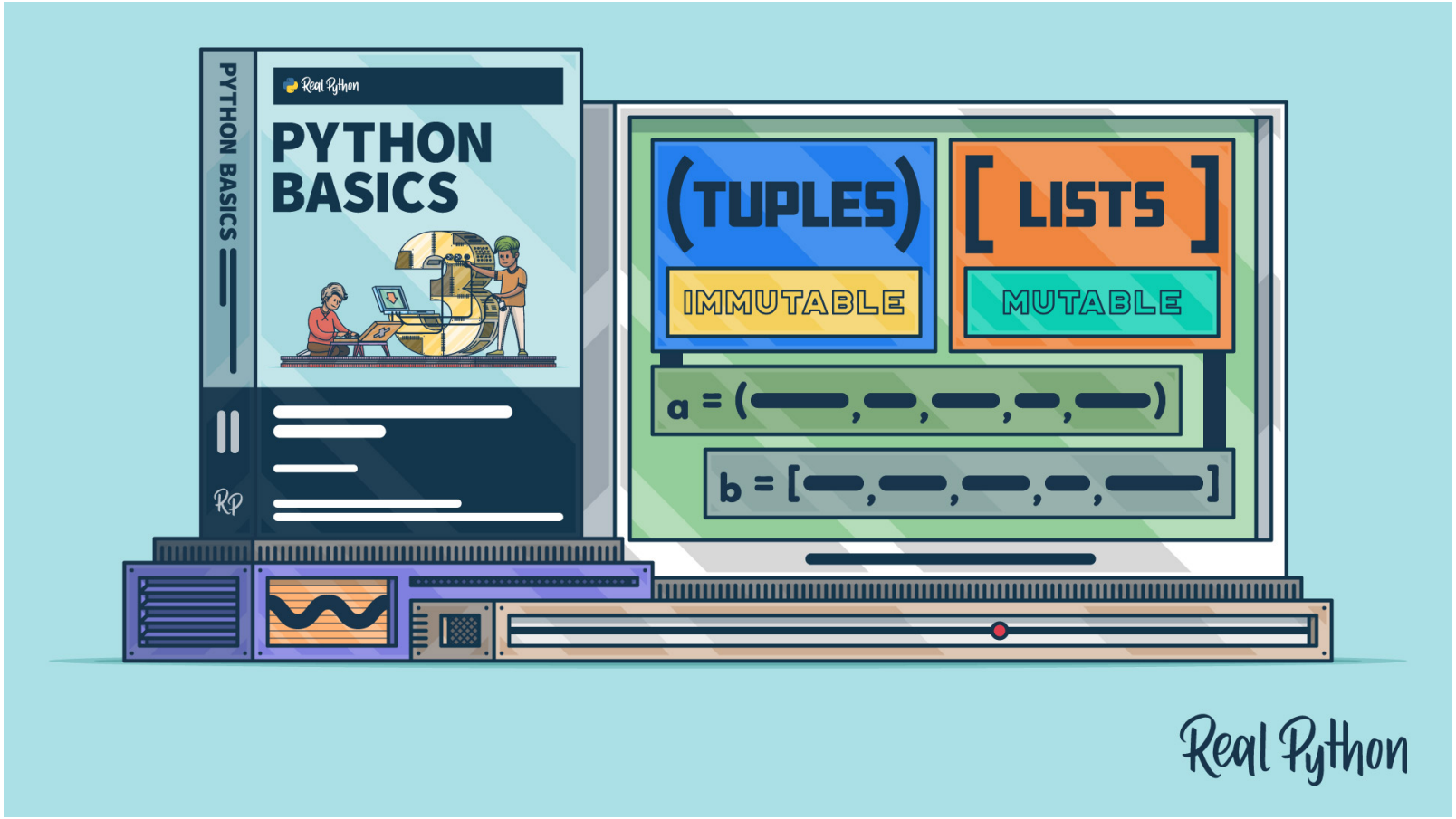
Python Basics - Tuples and Lists



Overview - Tuples and Lists

Fundamental data types:

- `str`
- `int`
- `float`

Overview - Tuples and Lists

- What if you want to describe a collection of elements?
- How do you combine several fundamental data types into something more complex?

Overview - Tuples and Lists

Lonely animals:

```
>>> "🐭"

>>> "🐻"

>>> "🦊"

>>> "🐷"
```

Overview - Tuples and Lists

Lonely ~~animals~~ elements:

```
>>> "🐭"

>>> "🐻"

>>> "🦊"

>>> "🐷"
```

Overview - Tuples and Lists

Hug them all together:

```
>>> ("🐭", "🐻", "🦊", "🐷")
```

Overview - Tuples and Lists

Hug them all together:

```
>>> ("🐭", "🐻", "🦊", "🐷")
```

Overview - Tuples and Lists

Certain **data structures** can model a collection of data:

- List of numbers
- Student names enrolled in a class
- Row in a spreadsheet
- Record in a database
- All your favorite animal emojis

Overview - Tuples and Lists

Certain **data structures** can model a collection of data:

- List of numbers
- Student names enrolled in a class
- Row in a spreadsheet
- Record in a database
- All your favorite animal emojis 🐶

Overview - Tuples and Lists

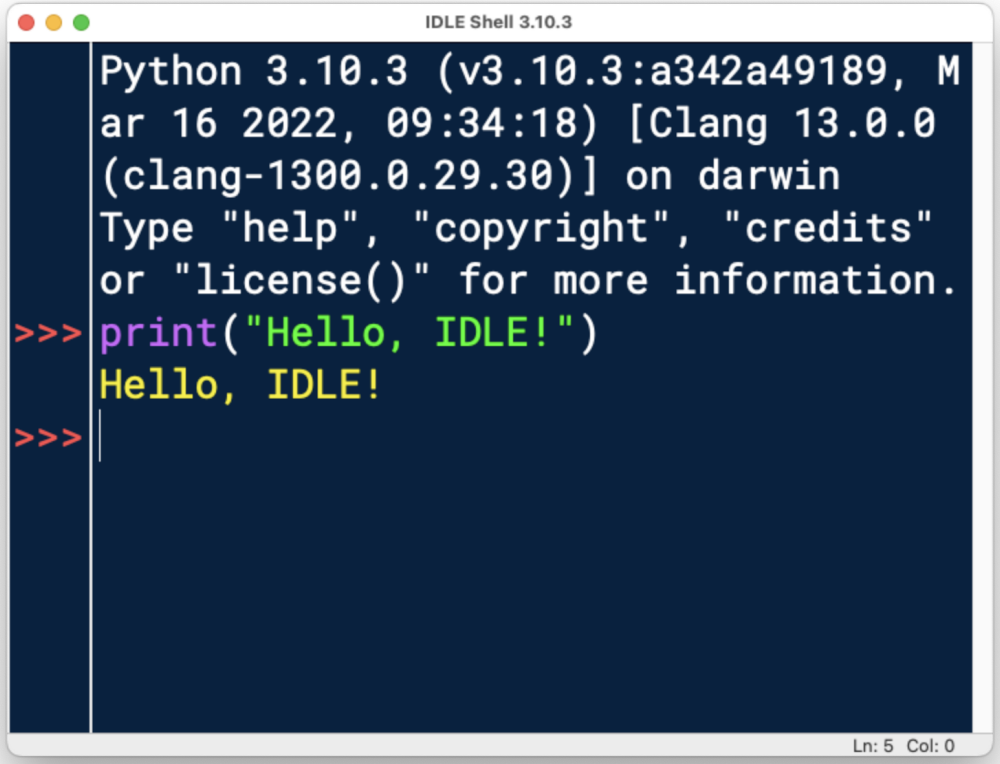
In this course you'll learn:

- How to work with **tuples** and **lists**
- When to use which of these two **collections**
- What **mutability** is, and why it's important to think about it

Table of Contents

Table of Contents

Background: Using IDLE



Additional Resources

Python Basics: Setting Up Python

Getting Started With Python IDLE

Overview - Tuples and Lists

Next Up: Tuples

Table of Contents

Table of Contents

Pronouncing Tuples

Pronunciation of Tuple:

- **Too-ple** (like Quadruple)
- **Tup-ple** (like Quintuple)

Pronouncing Tuples

Pronunciation of Tuple:

- **Too-ple** (like Quadruple)
- **Tup-ple** (like Quintuple)

Either is fine, and both are used.

What is a Tuple?

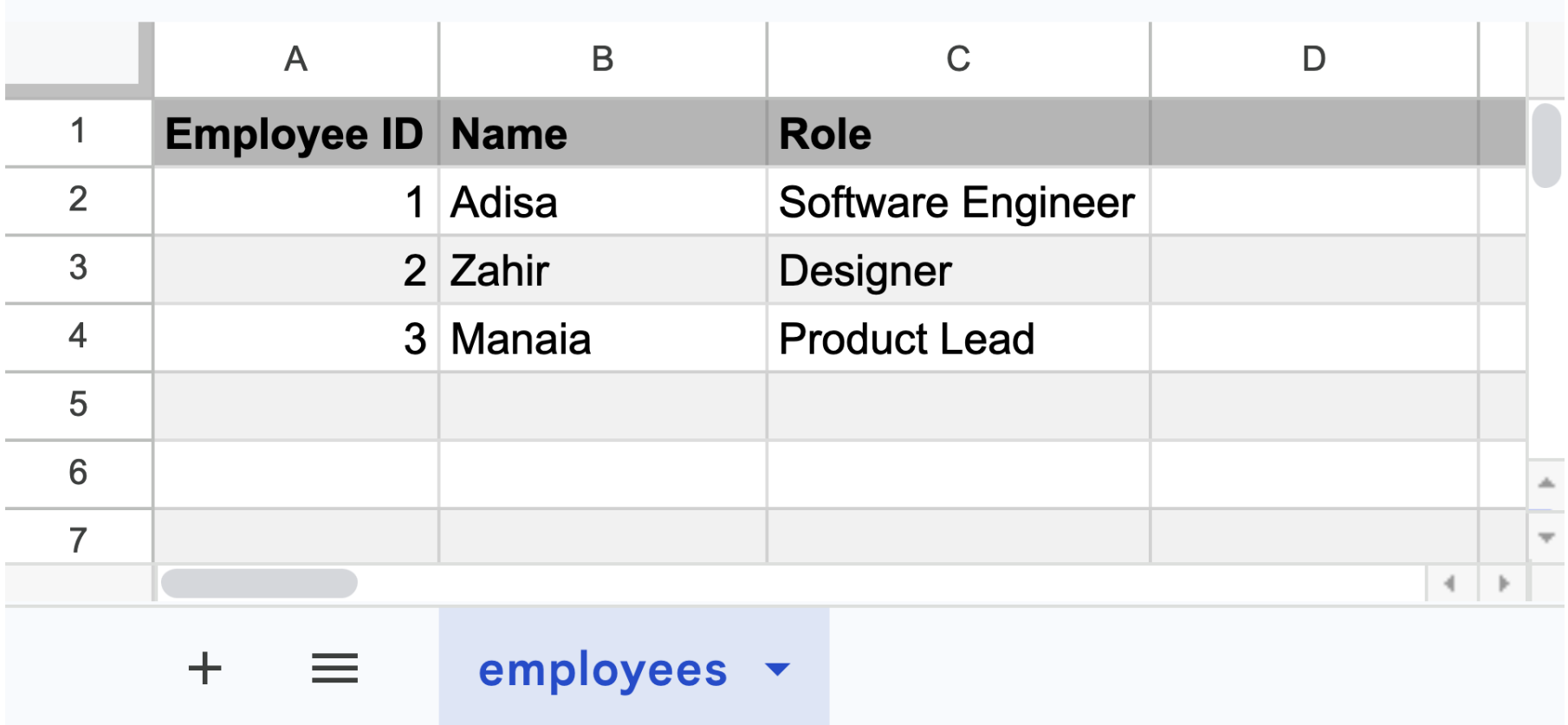
- A tuple is a **finite, ordered, immutable sequence** of values
- Python borrows the name and the notation from mathematics
    - Each element is separated by a comma
    - All elements are surrounded by a pair of parentheses
    - `(1, 2, 3)`

What is a Tuple?

What is a Tuple … Help?

Think About Database Records

… or rows in a spreadsheet:



	A	B	C	D
1	**Employee ID**	**Name**	**Role**	
2	1	Adisa	Software Engineer	
3	2	Zahir	Designer	
4	3	Manaia	Product Lead	
5				
6				
7				

employees

Think About Database Records

Employee ID	Name	Role
1	Adisa	Software Engineer
2	Zahir	Designer
3	Manaia	Product Lead

You could represent one record as a Python tuple:

```
(1, "Adisa", "Software Engineer")
```

What is a Tuple … Help?

A read-only database record

What is a Tuple … Help?

A read-only database record:

- Is a finite sequence of values
- Has a fixed structure
- Has fixed values
- Contains a variety of data types (text, numbers, …)
- Contains related data

What is a Tuple … Help?

A read-only database record

What is a Tuple?

Next Up: How to Create a Tuple

How to Create a Tuple

1. Tuple Literals
2. The built-in `tuple()`

Tuple Literals

String literal:

```
>>> 'Hello, World!'
>>> "1234"
```

Tuple Literals

Tuple literal:

- Comma-seperated values
- Surrounded by parentheses

Tuple Literals

Tuple literal:

- Comma-seperated values
- Surrounded by parentheses

```
>>> (1, 2, 3)
```

Tuple Literals

Tuple literal:

- Comma-seperated values
- Surrounded by parentheses

```
>>> (1, 2, 3)
>>> (1, "Adisa", "Software Engineer")
```

The Built-in tuple()

`tuple()` can be used to create a tuple from another sequence type

The Built-in tuple()

`tuple()` can be used to create a tuple from another sequence type

```
>>> tuple("Python")
('P', 'y', 't', 'h', 'o', 'n')
```

Defining Tuples

Next Up: Compare Tuples and Strings

Table of Contents

Similarities Between Tuples and Strings

- Tuples have a length
- Tuples support indexing and slicing
- Tuples are immutable
- Tuples are iterable

Similarities Between Tuples and Strings

Tuples have a length

```
>>> numbers = (1, 2, 3)
>>> len(numbers)
3
```

Similarities Between Tuples and Strings

Each element in a tuple has a numbered position called an **index**

- Access individual elements through their index
- Place the index number inside a pair of square brackets after the tuple

Similarities Between Tuples and Strings

Each element in a tuple has a numbered position called an **index**

- Access individual elements through their index
- Place the index number inside a pair of square brackets after the tuple

```
>>> numbers = (1, 2, 3)
>>> numbers[1]
2
```

Note: The index count starts with **zero**, like in most programming languages. Beware of **off-by-one errors**!

Similarities Between Tuples and Strings

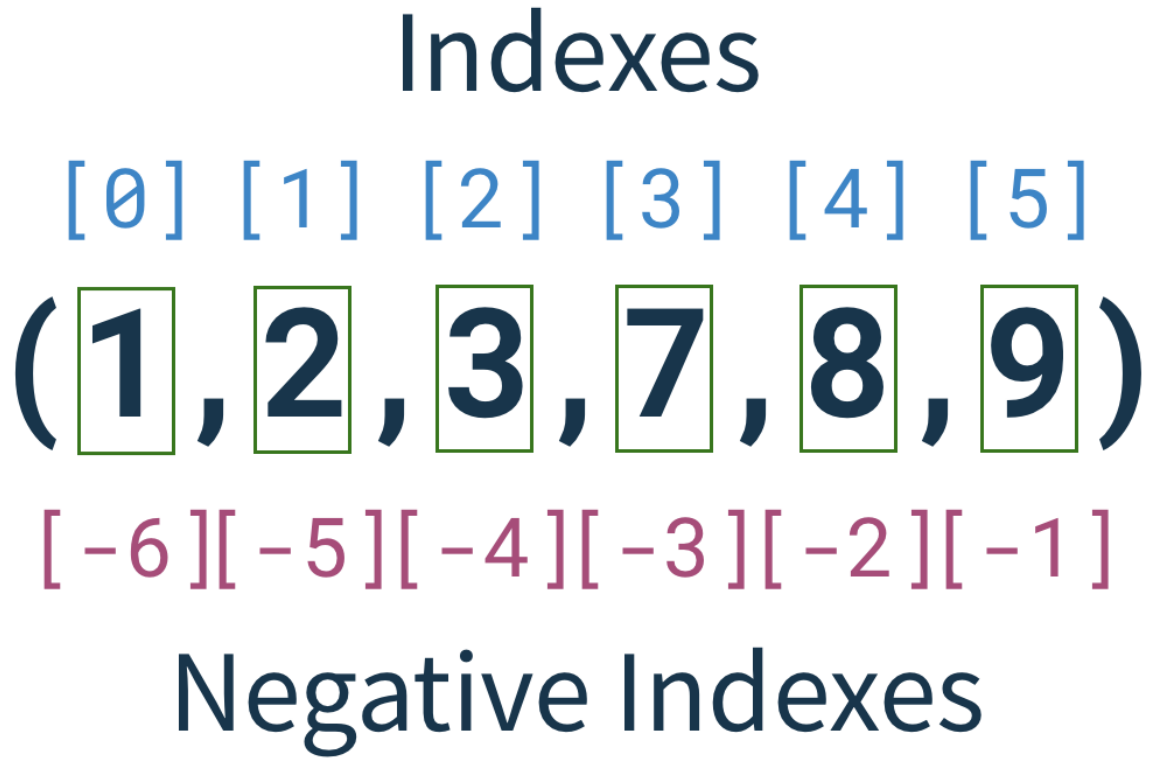
String Indexing

Similarities Between Tuples and Strings

Tuple Indexing

Similarities Between Tuples and Strings

Tuple Indexing



Similarities Between Tuples and Strings

Slicing

To extract a portion of a tuple, insert a colon between two index numbers: `[0:2]`

Similarities Between Tuples and Strings

Slicing

To extract a portion of a tuple, insert a colon between two index numbers:

```
>>> odd_numbers = (1, 3, 5, 7, 9)
>>> odd_numbers[2:4]
(5, 7)
```

- The slice `[x:y]` starts from the element at index `x`
- The slice goes up to, but does not include, the element at index `y`

Similarities Between Tuples and Strings

Tuples are immutable

You can’t change the value of an element in a tuple after creation

Similarities Between Tuples and Strings

Tuples are immutable

You can’t change the value of an element in a tuple after creation:

```
>>> numbers = (1, 2, 3)
>>> numbers[0] = 2
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    numbers[0] = 2
TypeError: 'tuple' object does not support item assignment
```

Similarities Between Tuples and Strings

Tuples are iterable:

```
>>> numbers = (1, 2, 3)
>>> for number in numbers:
...     print(number)
1
2
3
```

Table of Contents

Tuple Unpacking

Tuple Unpacking

You've seen that you can create a tuple by listing values separated by commas:

```
>>> rgb_lime_green = (50, 205, 50)

>>> rgb_lime_green
(50, 205, 50)

>>> type(rgb_lime_green)
<class 'tuple'>
```

Tuple Unpacking

You've seen that you can create a tuple by listing values separated by commas, where the parentheses are optional:

```
>>> rgb_lime_green = 50, 205, 50

>>> rgb_lime_green
(50, 205, 50)

>>> type(rgb_lime_green)
<class 'tuple'>
```

Tuple Unpacking

Similarly, you can assign elements of a tuple to multiple variables by **unpacking** it:

```
>>> red, green, blue = rgb_lime_green

>>> red
50
>>> green
205
>>> blue
50
```

Tuple Unpacking

With tuple unpacking, you can assign multiple variables in one line:

```
>>> employee_id, name, role = 1, "Adisa", "Software Engineer"

>>> employee_id
1
>>> name
'Adisa'
>>> role
'Software Engineer'
```

Tuple Unpacking

Note: Don't overdo it, only assign multiple variables via tuple unpacking if it makes your code more readable.

❌ Not well readable:

```
index, favorite_fruit, levels, is_defeated, attacker, number_of_lives, player_avatar = 1, "apple", [1, 2, 3], True, "dog", 9, "cat"
```

✅ Well readable:

```
latitude, longitude = 49.37, -123.37
```

Tuple Unpacking

Remember that tuples are like database records.

Python libraries that allow you to fetch data from databases may return rows as tuples.

You can unpack these if you need access to individual values.

Tuple Unpacking

```python
# setup.py
"""
Creates a database with an 'employees' table
and one record.
"""

import sqlite3

with sqlite3.connect("company.db") as connection:
    cursor = connection.cursor()
    cursor.execute("""
        CREATE TABLE employees (
            id INTEGER PRIMARY KEY,
            name TEXT,
            role TEXT
        )
    """)
    cursor.execute("""
        INSERT INTO employees (id, name, role)
        VALUES (?, ?, ?)
    """, (1, "Adisa", "Software Engineer"))
    connection.commit()
```

```python
# read.py
"""Reads and displays data from a SQLite database."""

import sqlite3

with sqlite3.connect("company.db") as connection:
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM employees")

    # Fetch one row - this will be a tuple
    row = cursor.fetchone()
    print(row)  # (1, 'Adisa', 'Software Engineer')
    print(type(row))  # <class 'tuple'>

    # Unpack the values
    employee_id, name, role = row
    print(name)  # Adisa
    print(role)  # Software Engineer
```

Tuple Unpacking

For unpacking to work, you the number of variables must match the number of values:

```
>>> a, b, c = 1, 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)

>>> a, b = 1, 2, 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

Table of Contents

Checking the Existence of Values With `in`

```
>>> vowels = ("a", "e", "i", "o", "u")

>>> "o" in vowels
True
>>> "x" in vowels
False
```

Table of Contents

Using Tuples to Return Multiple Values From a Function

```
>>> def get_stats(sequence):
...     return max(sequence), min(sequence)

>>> numbers = (1, 2, 3)
>>> get_stats(numbers)
(3, 1)

>>> maximum, minimum = get_stats(numbers)
>>> maximum
3
```

Table of Contents

Table of Contents

Review Exercises

1. Create a tuple literal named `location` that holds the floating point numbers `6.51`, `3.39`, and the strings `"Lagos"` and `"Nigeria"`, in that order.
2. Use index notation and `print()` to display the string at index `2` in `location`.
3. In a single line of code, unpack the values in `location` into four variables named `latitude`, `longitude`, `city`, and `country`. Then print each value on a separate line.

Review Exercises

1. Use `tuple()` and a string literal to create a tuple called `my_name` that contains the letters of your name.
2. Check whether the character `"x"` is in `my_name` using the `in` keyword.
3. Create a new tuple containing all but the first letter in `my_name` using slice notation.

Table of Contents

Table of Contents

What Is a List?

- A list is an **ordered, mutable sequence** of values
- Each element is separated by a comma
- All elements are surrounded by a pair of square brackets
- `[1, 2, 3]`

What Is a List?

It's another **sequence** type in Python

Lists are similar to tuples. Both:

- Can contain items of any data type
- Are Indexed by integers starting at `0`
- Support slice notation
- Support `in` to check for the existence of elements
- Are Iterable

What Is a List?

However, lists are *different* from tuples because lists are **mutable**.

You can change elements in a list after you've created it:

```
>>> numbers = [1, 2, 3]
>>> numbers[0] = "One"
>>> numbers
['One', 2, 3]
```

What Is a List?

Next Up: How to Create a List

How to Create a List

- List literal
  - `[1, 2, 3]`
- Built-in `list()`
  - `list(<sequence>)`
- String method `.split()`
  - `str.split(<separator>)`

How to Work With Lists

How to Work With Lists

Indexing:

```
>>> numbers = [1, 2, 3, 4]

>>> numbers[1]
2
```

How to Work With Lists

Slicing:

```
>>> numbers = [1, 2, 3, 4]

>>> numbers[1:3]
[2, 3]
```

How to Work With Lists

Membership testing:

```
>>> numbers = [1, 2, 3, 4]

>>> 2 in numbers
True

>>> "Bob" in numbers
False
```

How to Work With Lists

Iteration:

```
>>> numbers = [1, 2, 3, 4]

>>> for number in numbers:
...     if number % 2 == 0:
...         print(number)
...
2
4
```

How to Work With Lists

Next Up: Changing Elements in a List

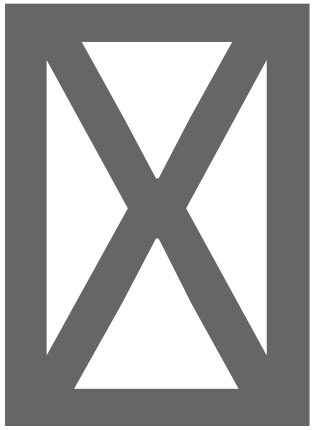
Table of Contents

Changing Elements in a List

You can change any element in a list after creation:

```
>>> numbers = [1, 2, 3]
>>> numbers[1] = 20
>>> numbers
[1, 20, 3]
```

Changing Elements in a List



Adding and Removing Elements

Adding and Removing Elements

Lists have useful methods that allow you to add and remove elements:

- `.insert(i, x)`
- `.append(x)`
- `.extend(<iterable>)`
- `.pop(i)`

Adding and Removing Elements

Use `.insert()` to insert an element at an index position:

```
>>> numbers = [1, 3]

>>> numbers.insert(1, "two")  # (index_position, element)
>>> numbers
[1, "two", 3]
```

Adding and Removing Elements

Use `.append()` to add an element to the end of the list:

```
>>> numbers = [1, 2]

>>> numbers.append(3)
>>> numbers
[1, 2, 3]
```

Adding and Removing Elements

Use `.extend()` to add a collection to the end of the list:

```
>>> numbers = [1, 2]

>>> numbers.extend([3, 4])
>>> numbers
[1, 2, 3, 4]

>>> numbers.extend((5, 6))
>>> numbers
[1, 2, 3, 4, 5, 6]
```

Adding and Removing Elements

Use `.pop()` to remove an element from the list:

```
>>> numbers = [1, 2, 3, 4]

>>> numbers.pop(2)  # Index position
3
>>> numbers
[1, 2, 4]

>>> numbers.pop()  # Default: last element
4
>>> numbers
[1, 2]
```

Adding and Removing Elements

Breathe…

Adding and Removing Elements

Breathe… Let’s IDLE!

Working With Lists of Numbers

Working With Lists of Numbers

You can perform mathematical operations on collections using built-in functions:

- `sum()`
- `min()`
- `max()`

Working With Lists of Numbers

Next Up: Creating List Comprehensions

Creating List Comprehensions

Creating List Comprehensions

A list comprehensions is a shorthand for a `for` loop:

Creating List Comprehensions

A list comprehensions is a shorthand for a `for` loop:

```
>>> numbers = (1, 2, 3)

>>> squares = []
>>> for num in numbers:
>>>     squares.append(num**2)
...

>>> squares
[1, 4, 9]
```

Creating List Comprehensions

A list comprehensions is a shorthand for a `for` loop:

```
>>> numbers = (1, 2, 3)

>>> squares = [num**2 for num in numbers]

>>> squares
[1, 4, 9]
```

Creating List Comprehensions

A list comprehensions is a shorthand for a `for` loop:

```
>>> numbers = (1, 2, 3)

>>> squares = [num**2 for num in numbers]

>>> squares
[1, 4, 9]

>>> type(squares)
<class 'list'>
```

Creating List Comprehensions

Next Up: Nesting and Copying Lists

Review Exercises

1. Create a list named `food` with two elements, `"rice"` and `"beans"`.
2. Append the string `"broccoli"` to `food` using `.append()`.
3. Add the strings `"bread"` and `"pizza"` to `food` using `.extend()`.
4. Print the first two items in the `food` list using `print()` and slice notation.
5. Print the last item in `food` using `print()` and index notation.

Review Exercises

1. Create a list called `breakfast` from the string `"eggs, fruit, orange juice"` using the string `.split()` method.
2. Verify that `breakfast` has three items using `len()`.
3. Create a new list called `lengths` using a list comprehension that contains the lengths of each string in the `breakfast` list.

Table of Contents

Nesting Lists

You can **nest** lists and tuples:

```
>>> maori_words = [["whānau", "family"],
...                ["kai", "food"],
...                ["aroha", "love"]]
```

Nesting Lists

You can access elements using **double index notation**:

```
>>> maori_words = [["whānau", "family"],
...                ["kai", "food"],
...                ["aroha", "love"]]

>>> maori_words[2][0]
'aroha'
>>> maori_words[2][1]
'love'
```

Nesting Lists

You can nest lists as deeply as you want to:

```
deeply_nested_list = [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[["Help!"]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
```

But…

Nesting Lists

You can nest lists as deeply as you want to:

```
deeply_nested_list = [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[["Help!"]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]
```

But… Don’t do it.

Nesting Lists

Next Up: Copying Lists

Copying Lists

Copying Lists

Variable Names are references to objects in memory:

```
>>> matrix = [[1, 2], [3, 4]]

>>> the_matrix = matrix
>>> the_matrix[-1] = ["Neo", "Trinity"]

>>> the_matrix
[[1, 2], ['Neo', 'Trinity']]
>>> matrix
[[1, 2], ['Neo', 'Trinity']]
```

Copying Lists

You can create a **shallow copy** using the `[:]` slice notation:

```
>>> matrix = [[1, 2], [3, 4]]

>>> the_matrix = matrix[:]
>>> the_matrix[-1] = ["Neo", "Trinity"]

>>> the_matrix
[[1, 2], ['Neo', 'Trinity']]
>>> matrix
[[1, 2], [3, 4]]
```

Copying Lists

Shallow copies contain copied references:

```
>>> matrix = [[1, 2], [3, 4]]

>>> the_matrix = matrix[:]
>>> the_matrix[0][0] = "Morpheus"

>>> the_matrix
[["Morpheus", 2], [3, 4]]
>>> matrix
[["Morpheus", 2], [3, 4]]
```

Copying Lists

Only a **deep copy** creates a truly independent copy of an object:

```
>>> import copy

>>> matrix = [[1, 2], [3, 4]]
>>> the_matrix = copy.deepcopy(matrix)
>>> the_matrix[0][0] = "The Oracle"

>>> the_matrix
[["The Oracle", 2], [3, 4]]
>>> matrix
[[1, 2], [3, 4]]
```

Copying Lists

Next Up: Sorting Lists

Sorting Lists

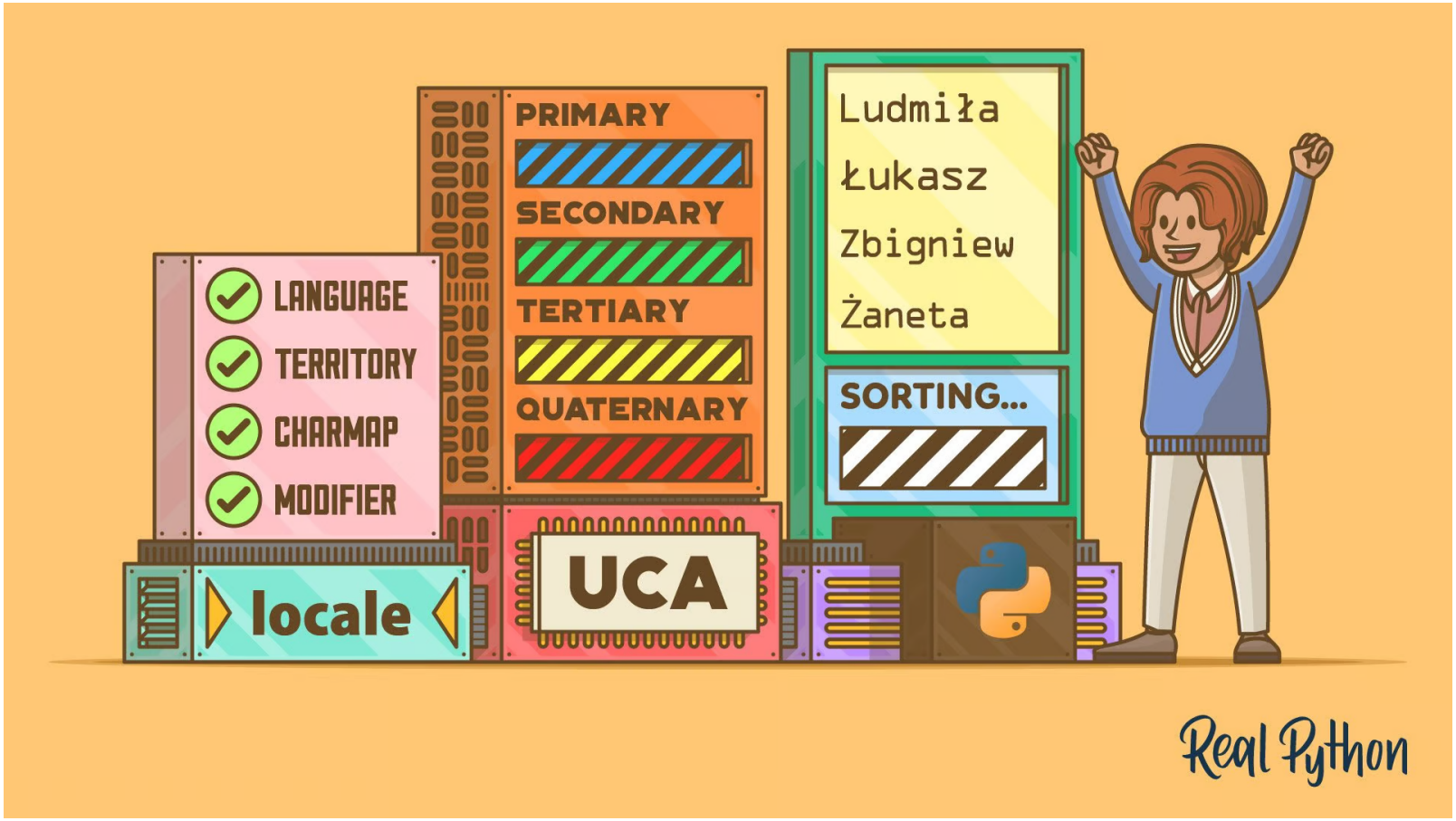
Sorting Lists

You can sort lists using `.sort()`:

```
>>> colors = ["red", "yellow", "green", "blue"]
>>> colors.sort()
>>> colors
['blue', 'green', 'red', 'yellow']

>>> numbers = [1, 10, 5, 3]
>>> numbers.sort()
>>> numbers
[1, 3, 5, 10]
```

Python sorts strings alphabetically (according to their Unicode code point values), and numbers ascending.

How to Sort Unicode Strings Alphabetically in Python



Sorting Lists

You can sort lists in reverse using `reverse=True`:

```
>>> numbers = [1, 10, 5, 3]
>>> numbers.sort(reverse=True)

>>> numbers
[10, 5, 3, 1]
```

This approach inverts the results of using `.sort()` without setting `reverse` to `True`.

Sorting Lists

The `.sort()` method accepts a function as an argument to `key`. You can use it customize how to sort even further:

```
>>> colors = ["red", "yellow", "green", "blue"]

>>> colors.sort(key=len)

>>> colors
['red', 'blue', 'green', 'yellow']
```

In this example, you used the built-in `len` function to sort the elements ascending by their length.

Sorting Lists

The `.sort()` method accepts a function as an argument to `key`. You can use it customize how to sort even further.

Note: Pass the function object without calling it.

Note: The function that you pass to `key` must accept only a single argument.

Sorting Lists

You can pass user-defined functions to `key`:

```
>>> def get_second_element(item):
...     return item[1]

>>> items = [(-10, 2), (0, 3), (10, 1)]

>>> items.sort(key=get_second_element)
>>> items
[(10, 1), (-10, 2), (0, 3)]
```

In this example, you sorted the list ascending by the second integer values in each of the tuples that are elements of the `items` list.

Table of Contents

Table of Contents

Table of Contents

Review Exercises

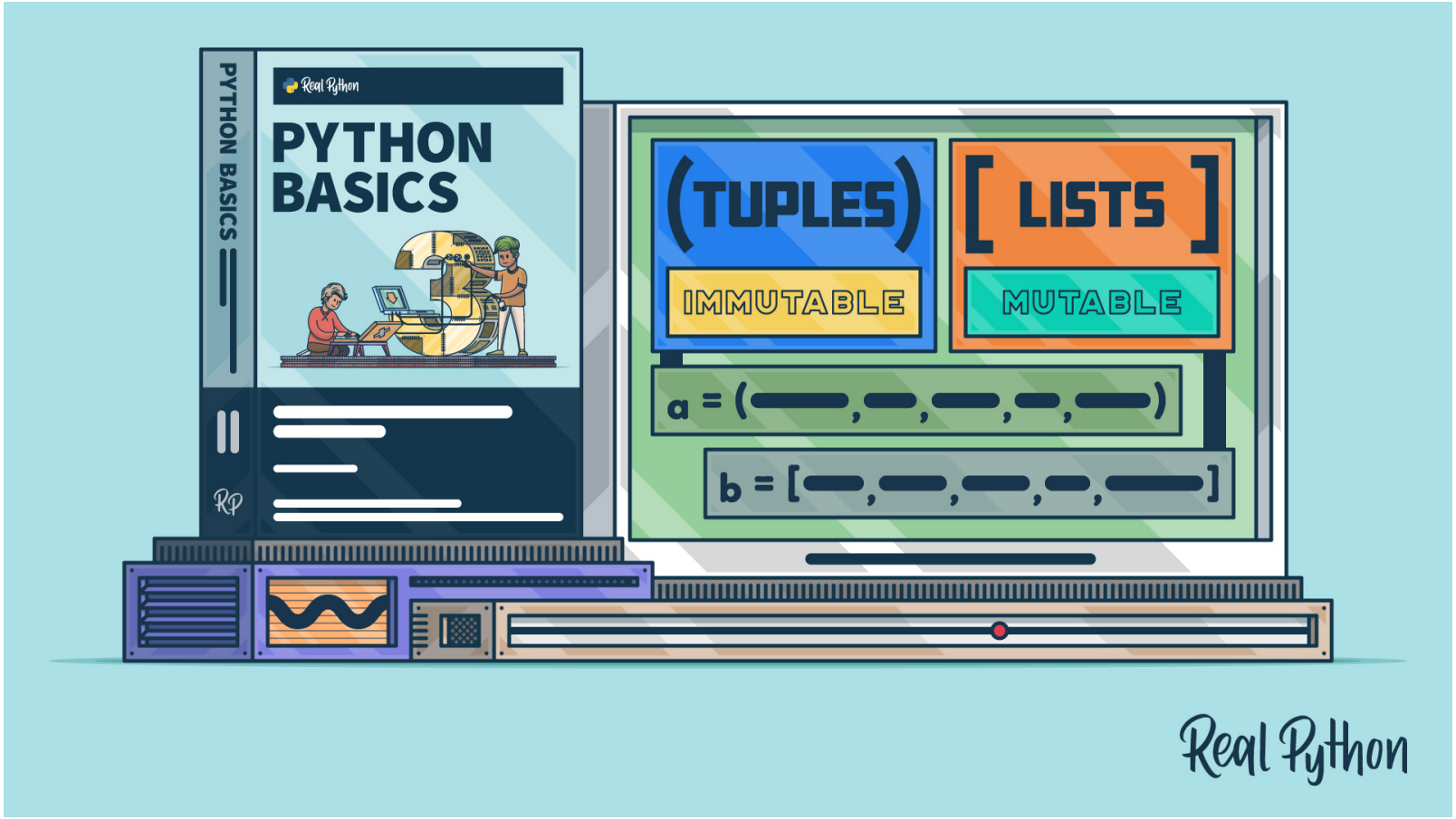
1. Create a tuple called `data` with two values. The first value should be the tuple `(1, 2)`, and the second value should be the tuple `(3, 4)`.
2. Write a `for` loop that loops over `data` and prints the sum of each nested tuple. The output should look like this:

```
Row 1 sum: 3
Row 2 sum: 7
```

Review Exercises

1. Create the list `[4, 3, 2, 1]` and assign it to the variable `numbers`.
2. Create a copy of the `numbers` list using the `[:]` slice notation.
3. Sort the `numbers` list in numerical order using `.sort()`.

Python Basics - Tuples and Lists



Summary

Tuples: Immutable Sequences

- Think of Tuples as Rows
- Create Tuples
- Compare Tuples and Strings
- Unpack Tuples
- Check the Existence of Values With `in`
- Return Multiple Values From a Function

```
>>> (1, "two", 3.0)
(1, 'two', 3.0)

>>> 1, 2, 3
(1, 2, 3)

>>> name, age = "Jo", 73
>>> name
'Jo'
>>> age
73
```

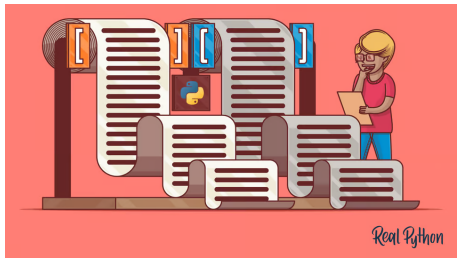
Summary

Lists: Mutable Sequences

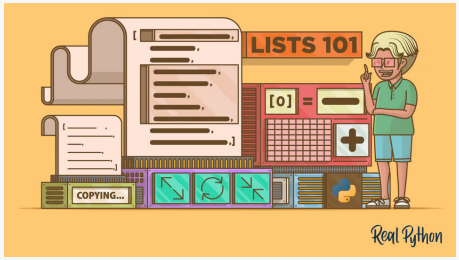
- Create Lists
- Work With Lists
- Change Elements in a List
- Add and Remove Elements
- Work With Lists of Numbers
- Create List Comprehensions
- Nest and Copy Lists
- Sort Lists

```
>>> numbers = [1, 2, 3]
>>> numbers
[1, 2, 3]

>>> numbers[1] = 100
>>> numbers
[1, 100, 3]

>>> [num**2 for num in numbers]
[1, 10000, 9]
```

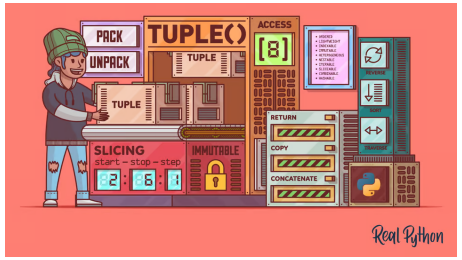
Additional Resources



Lists and Tuples in Python

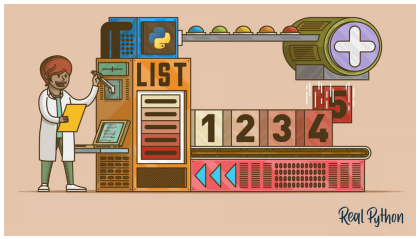


Python's `list` Data Type: A Deep Dive With Examples



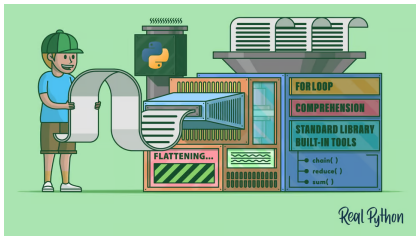
Python's `tuple` Data Type: A Deep Dive With Examples

Additional Resources



Python's `.append()`: Add Items to Your Lists in Place

Reverse Python Lists: Beyond `.reverse()` and `reversed()`



How to Flatten a List of Lists in Python

Python Basics - Tuples and Lists