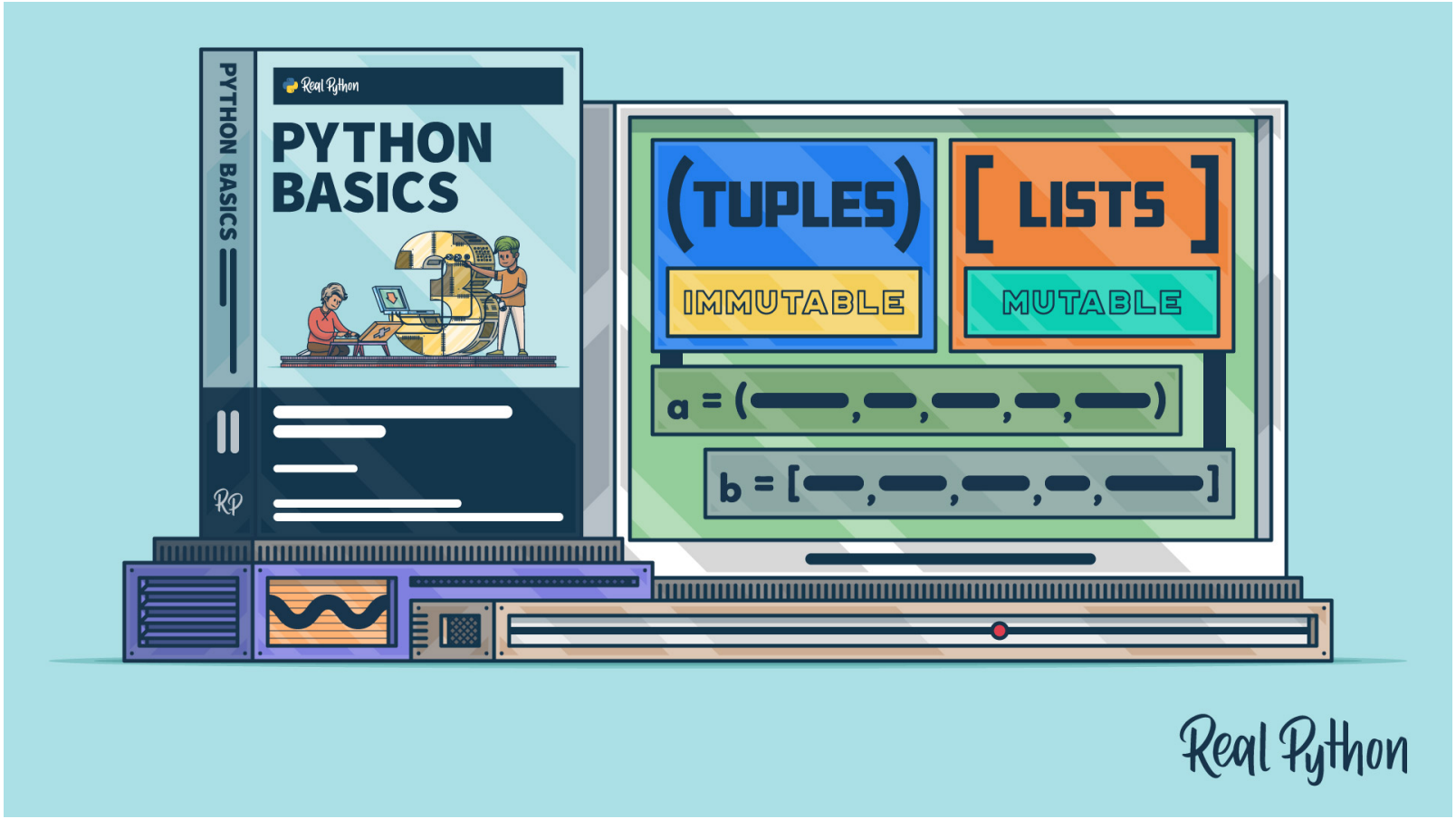
Python Basics Exercises: Tuples and Lists

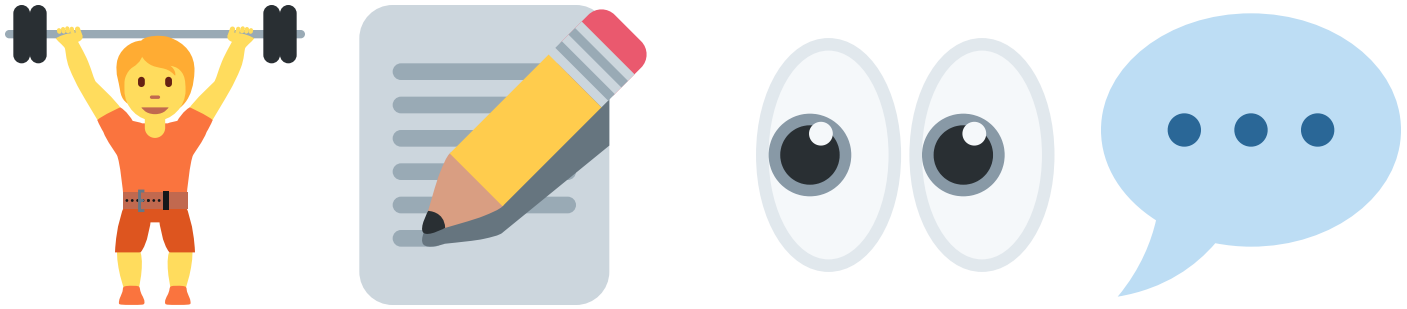


Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course

Real Python Exercises Course



Real Python Exercises Course

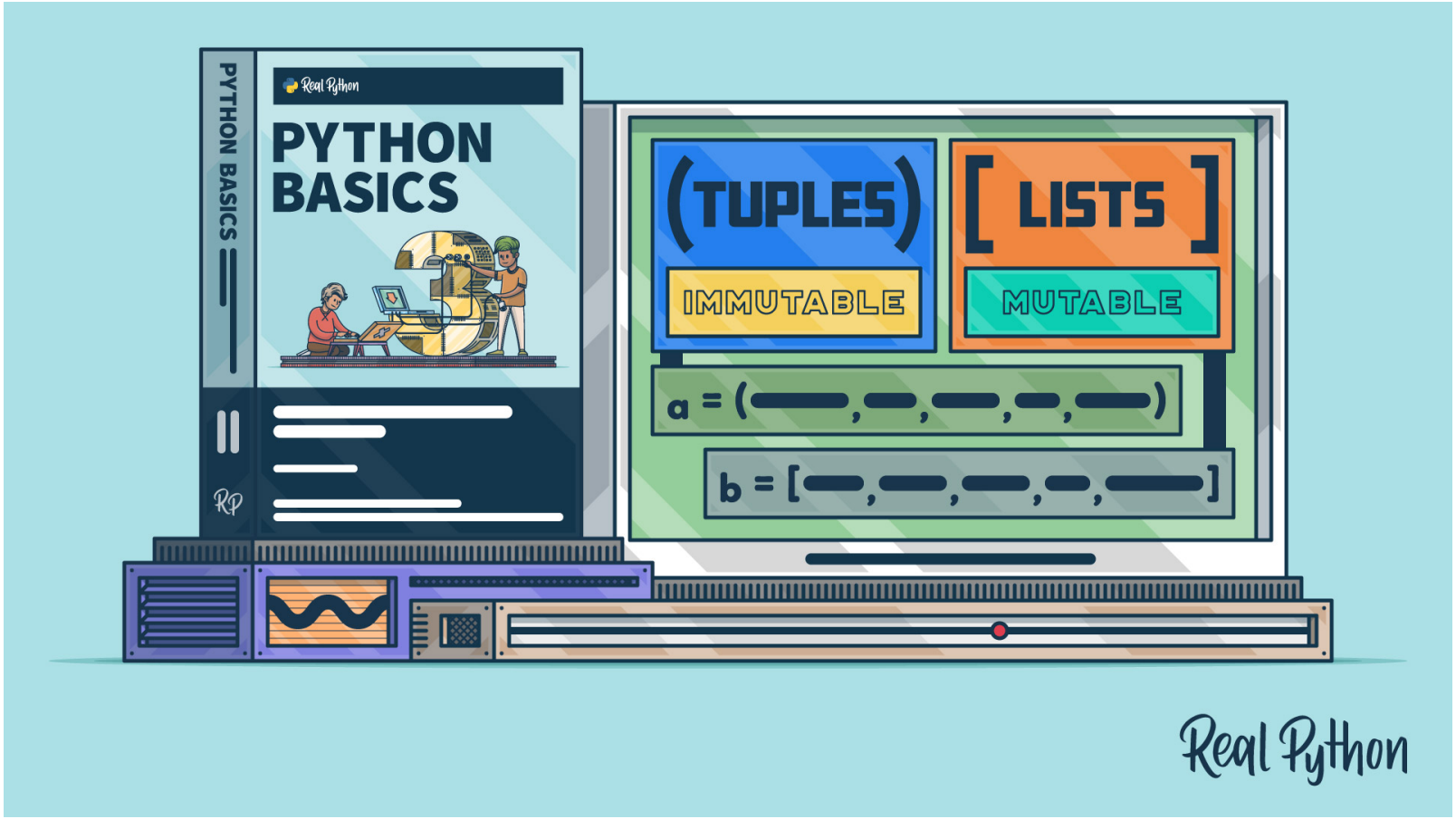
The three steps for each task:

1. 👀 Learn about the exercise
2. 👨‍💻 Code your solution
3. ↔️ Compare your solution

Python Basics Exercises: Tuples and Lists

1. Review Exercises
2. Challenge

Background - Python Basics: Tuples and Lists

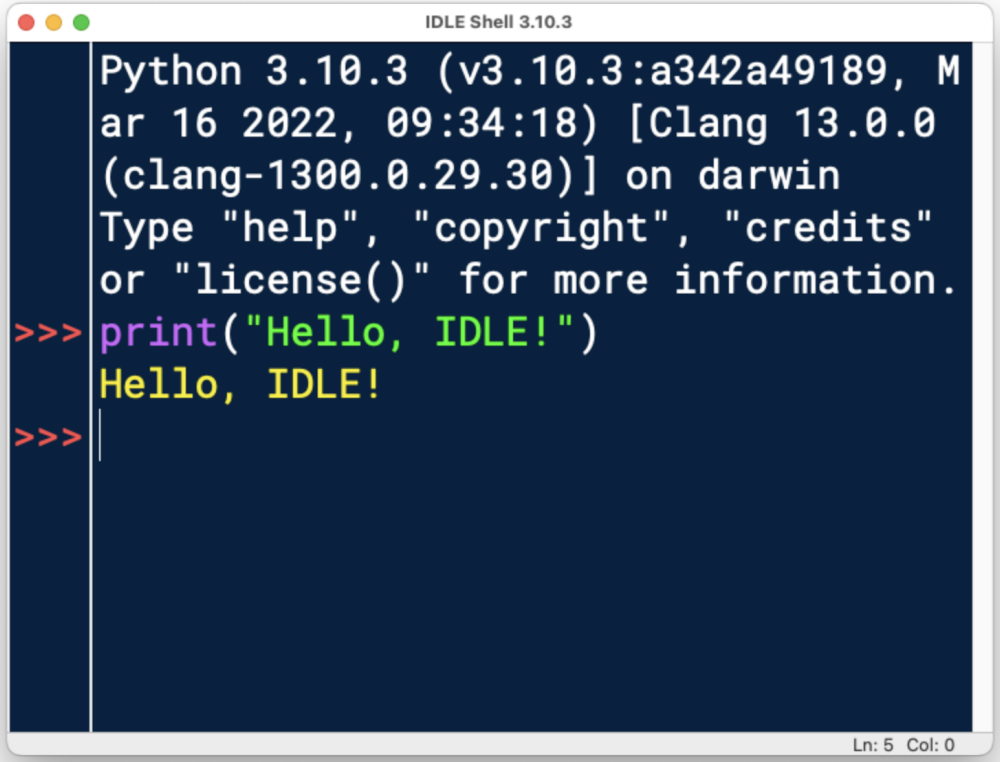


Background - Python Basics: Tuples and Lists

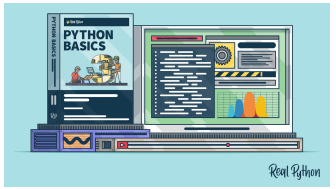
In this course, you'll **practice**:

- Tuples
- Lists
- Indexing
- Slicing
- Tuple unpacking
- List comprehensions
- Mutability
- Conditional logic and loops

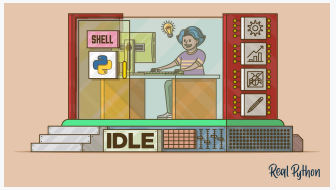
Background - Using IDLE



Additional Resources



Python Basics: Setting Up Python



Getting Started With Python IDLE

Ready to Get Started?

```
>>> animals = ("🐭", "🐻", "🦊", "🐷")

>>> [f"{animal}-<(Yes!)" for animal in animals]
```

Ready to Get Started?

```
>>> animals = ("🐭", "🐻", "🦊", "🐷")

>>> [f"{animal}-<(Yes!)" for animal in animals]

['🐭-<(Yes!)', '🐻-<(Yes!)', '🦊-<(Yes!)', '🐷-<(Yes!)']
```

🤔 Find a Location (Exercise)

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 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.

💡 Find a Location (Solution)

```
>>> location = (6.51, 3.39, "Lagos", "Nigeria")

>>> location[2]
'Lagos'

>>> latitude, longitude, city, country = location
>>> latitude
6.51
>>> longitude
3.39
>>> city
'Lagos'
>>> country
'Nigeria'
```

🤔 Person X (Exercise)

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.

💡 Person X (Solution)

Note that if you enter your own name, then the results will look differently.

```
>>> my_name = tuple("Martin")

>>> "x" in my_name
False

>>> my_name[1:]
('a', 'r', 't', 'i', 'n')
```

🤔 Shopping List (Exercise)

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. Display the first two items in the `food` list using slice notation.
5. Display the last item in `food` using index notation.

💡 Shopping List (Solution)

```
>>> food = ["rice", "beans"]

>>> food.append("broccoli")

>>> food.extend(["bread", "pizza"])

>>> food[:2]  # Or: food[0:2]
['rice', 'beans']

>>> food[-1]
'pizza'
```

🤔 Long Breakfast (Exercise)

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.

Long Breakfast (Solution)

```
>>> breakfast = "eggs, fruit, orange juice".split(", ")

>>> len(breakfast) == 3
True

>>> lengths = [len(item) for item in breakfast]
>>> lengths
[4, 5, 12]
```

🤔 Tuple Sums (Exercise)

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
```

Tuple Sums (Solution)

```
>>> data = ((1, 2), (3, 4))

>>> index = 1
>>> for row in data:
...     print(f"Row {index} sum: {sum(row)}")
...     index += 1
Row 1 sum: 3
Row 2 sum: 7
```

🤔 Sorted Numbers Copy (Exercise)

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 copy of the `numbers` list in numerical order using `.sort()`.
4. Display both `numbers` and its copy to proof that they look differently.

💡 Sorted Numbers Copy (Solution)

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

>>> numbers_copy = numbers[:]

>>> numbers_copy.sort()

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

🤔 Challenge: Poetry Generator

In this challenge, you'll write a program that generates poetry.

Create five lists for different word types:

```
nouns = ["badger", "horse", "aardvark", "mouse", "gorilla", "elephant", "eagle", "sparrow"]
verbs = ["hugs", "bounces", "meows", "hauls", "whispers", "flutters", "gallops", "shimmers"]
adjectives = ["furry", "incredulous", "fragrant", "exuberant", "glistening", "melancholic", "serene"]
prepositions = ["against", "after", "into", "beneath", "upon", "for", "in", "like", "over", "within"]
adverbs = ["curiously", "extravagantly", "graciously", "reluctantly", "meticulously", "vigorously"]
```

Feel free to copy these suggested words, or use different ones.

🤔 Challenge: Poetry Generator

Randomly select the following number of elements from each list:

- Three nouns
- Three verbs
- Three adjectives
- Two prepositions
- One adverb

You can do this with the `choice()` function in the `random` module. This function takes a list as input and returns a randomly selected element of the list.

🤔 Challenge: Poetry Generator

For example, here's how you use `random.choice()` to get random element from the list `["a", "b", "c"]`:

```
import random

random_element = random.choice(["a", "b", "c"])
```

Note: Keep exploring other options as well. Maybe you can find another function in `random` that could be helpful. Maybe you can use a list method that you haven't used before.

🤔 Challenge: Poetry Generator

Using the randomly selected words, generate and display a poem with the following structure inspired by Clifford Pickover:

```
{article} {adj1} {noun1}

{article} {adj1} {noun1} {verb1} {prep1} the {adj2} {noun2}
{adverb1}, the {noun1} {verb2}
the {noun2} {verb3} {prep2} the {adj3} {noun3}
```

Here, `adj` stands for *adjective* and `prep` for *preposition*. Note that the `article` will need to be either `A` or `An` depending on the first letter of the next word.

🤔 Challenge: Poetry Generator

Here's an example of the kind of poem your program might generate:

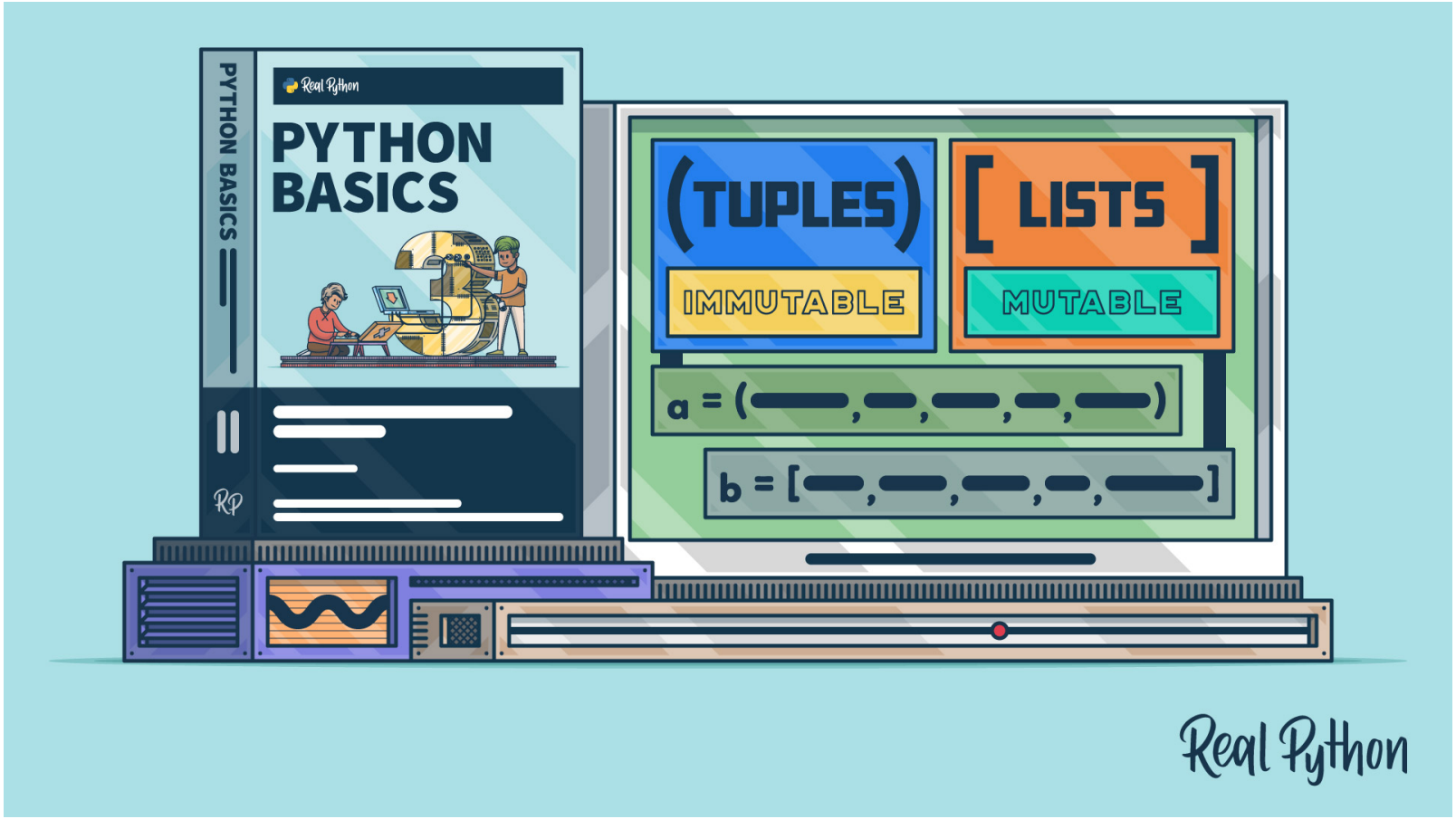
```
A melancholic aardvark

A melancholic aardvark shimmers like the furry horse
extravagantly, the aardvark gallops
the horse meows over the incredulous gorilla
```

Each time your program runs, it should generate a new poem.

Challenge: Poetry Generator

Download the code from *Materials* for the possible solution that you saw in the video course.

Python Basics Exercises: Tuples and Lists



Summary and Additional Resources

```
>>> animals = ("🐭", "🐻", "🦊", "🐷")

>>> [f"{animal}-<(Yay!)" for animal in animals]

['🐭-<(Yay!)', '🐻-<(Yay!)', '🦊-<(Yay!)', '🐷-<(Yay!)']
```

Summary and Additional Resources

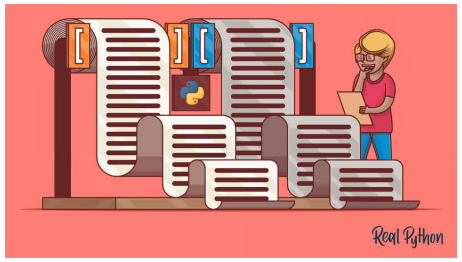
In this course, you **practiced** using:

- Tuples
- Lists
- Indexing
- Slicing
- Tuple unpacking
- List comprehensions
- Mutability
- Conditional logic and loops

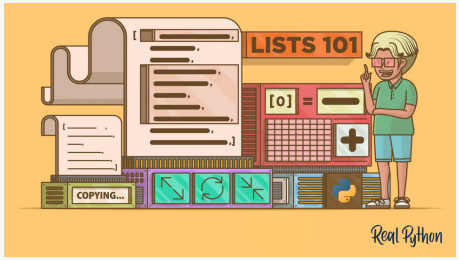
Tips

- 💬 Use code comments to help you get organized
- 🧩 Break exercises into smaller tasks
- 🧪 Try different approaches to solve a task
- 🔄 Refactor your code
- 🏷️ Use descriptive variable names
- ✅ Test repeatedly to see whether the code does what you expect it to do

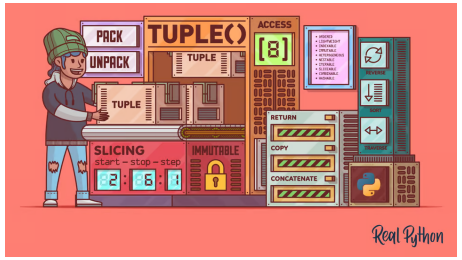
Additional Resources



Lists and Tuples in Python

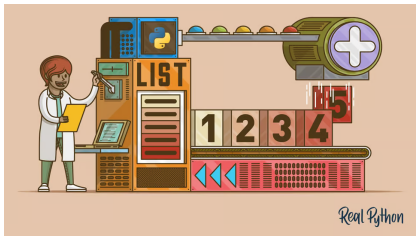


Python's `list` Data Type: A Deep Dive With Examples



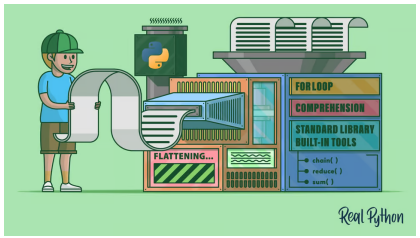
Python's `tuple` Data Type: A Deep Dive With Examples

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

Congratulations and Thanks!