Compare lists, strings, and tuples

You've now learned about some of Python's core iterable sequence data structures, including strings, lists, and tuples. These structures share many similarities, but there are some key differences between them. Data professionals must often decide which data structures work best to solve a particular problem, so understanding the relationship between these classes can help you make informed decisions in your work. This reading is a guide to the similarities and differences between strings, lists, and tuples.

Strings

Syntax/instantiation

Note: The following code block is not interactive.

- Single, double, or triple quotes:

```
my_string3 = """
marathon
golfcart
"""
empty_str = ''
my_string1 = 'minerals'
my_string2 = "martin"
```

Note: Using triple quotes to write a string over multiple lines will insert newlines (`\n`).

```
my_string3 = """
marathon
golfcart
"""

my_string3
```

RunReset

```
marathon
golfcart
```

- The **`str()`** function can be used for instantiation and conversion.

Note: The following code block is not interactive.

```
 empty_str = str()
my_string = str(125)
```

Content

- Strings can contain any character—letters, numbers, punctuation marks, spaces—but everything between the opening and closing quotation marks is part of the same single string.

Mutability

- Strings are **immutable**. This means that once a string is created, it cannot be modified. Any operation that appears to modify a string actually creates a new string object.

Usage

- Strings are most commonly used to represent text data.

Methods

The Python **`string`** class comes packed with many useful methods to manipulate the data contained in strings. For more information on these methods, refer to Common String Operations in the Python documentation.

Lists

Syntax/instantiation

- Brackets, with each element separated by a comma:

Note: The following code block is not interactive.

```
empty_list = []
my_list = [1, 2, 3, 4, 5]
```

- The `list()` function can be used for instantiation and conversion. Note that this function only works on iterable data types.

```
print(list('rocks'))
print(list(('stones', 'water', 'underground')))
```

RunReset

```
['r', 'o', 'c', 'k', 's']
['stones', 'water', 'underground']
```

Content

- Lists can contain any data type, and in any combination. So, a single list can contain strings, integers, floats, tuples, dictionaries, and other lists.

Note: The following code block is not interactive.

```
my_list = [1, 2, 1, 2, 'And through', ['and', 'through']]
```

Mutability

- Lists are **mutable**. This means that they can be modified after they are created.

```
num_list = [1, 2, 3]
num_list[0] = 5446
print(num_list)
```

RunReset

```
[5446, 2, 3]
```

Usage

- Lists are very versatile and therefore are used in numerous cases. Some common ones are:
    - Storing collections of related items
    - Storing collections of items that you want to iterate over: Because lists are ordered, you can easily iterate over their elements using a for loop or list comprehension.
    - Sorting and searching: Lists can be sorted and searched, making them useful for tasks such as finding the minimum or maximum value in a list or sorting a list of items alphabetically.
    - Modifying existing data: Because lists are mutable, they are useful for situations in which you know you'll need to modify your data.

    - Storing results: Lists can be used to store the results of a computation or a series of operations, making them useful in many different programming tasks.

Methods

- You can find methods for the Python list class in More on Lists in the Python documentation.

Tuples

Syntax/instantiation

- Parentheses, with each element separated by a comma:

Note: The following code block is not interactive.

```
empty_tuple = ()
my_tuple = (1, 'z')
```

Note: When using parentheses to declare a tuple with just a single element, you must use a trailing comma.

```
test1 = (1)
test2 = (2,)

print(type(test1))
print(type(test2))
```

RunReset

```
<class 'int'>
<class 'tuple'>
```

- No parentheses, but each element followed by a comma (even if there's only one element):

```
tuple1 = 1,
tuple2 = 2, 3
```

```
print(type(tuple1))
print(type(tuple2))
```

RunReset

```
<class 'tuple'>
<class 'tuple'>
```

- The **`tuple()`** function can be used for instantiation, and for conversion of iterable data types.

Note: The following code block is not interactive.

```
empty_tuple = tuple()
my_tuple = tuple([1, 'z'])
```

Content

- Tuples can contain any data type, and in any combination. So, a single tuple can contain strings, integers, floats, lists, dictionaries, and other tuples.

Note: The following code block is not interactive.

```
my_tuple = (1871, 'all', 'mimsy', ('were', 'the'), ['borogroves'])
```

Mutability

- Tuples are **immutable**. This means that once a tuple is created, it cannot be modified.

Usage

- Common uses of tuples include:
  - Returning multiple values from a function
  - Packing and unpacking sequences: You can use tuples to assign multiple values in a single line of code.
  - Dictionary keys: Because tuples are immutable, they can be used as dictionary keys, whereas lists cannot. (You'll learn more about dictionaries later.)
  - Data integrity: Due to their immutable nature, tuples are a more secure way of storing data because they safeguard against accidental changes.

Methods

- Because tuples are built for data security, Python has only two methods that can be used on them:
  - **`count()`** returns the number of times a specified value occurs in the tuple.
  - **`index()`** searches the tuple for a specified value and returns the index of the first occurrence of the value.

Key takeaways

Strings, lists, and tuples are all iterable sequential data structures that share many similarities. They also have fundamental differences that you should be aware of so you can make effective choices in your work as a data professional. When selecting a data structure, consider its manner of instantiation, content, mutability, and the use case.

Resources:

- For more information about strings, refer to the Introduction to Python strings documentation.
- For more information about lists, refer to the Introduction to Python lists documentation.
- For more information about tuples, refer to the Python Standard Data Types tuples documentation.