|  |  |
| --- | --- |
| Lab 8: Tuples and Strings | Name: Lex Baker  Date: 10/11/22 |

Thus far we have seen a one-to-one correspondence between a single variable name and a single value. It is common and incredibly useful to put a single variable name in correspondence with multiple, related values. In Python, a single variable that corresponds to an **ordered** set of values is referred to as a **sequence** (most often called an **array** in other languages). This Lab will introduce you to **tuples**, the simplest and most flexible sequences, and show you that **strings can also be thought of as sequences**.

**1.** **Creating tuples –** The Mariners are off to a good start. The table below shows the scores of their first 10 games.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Game # | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| M’s | 10 | 7 | 8 | 2 | 3 | 3 | 5 | 0 | 6 | 1 |
| Opponents | 3 | 5 | 2 | 3 | 1 | 6 | 3 | 2 | 4 | 3 |

**a.** Enter the following statement in the Python command line to create a tuple that contains the runs scored by the Mariners in each of their first 10 games. The value(s) of this tuple are then displayed with a call to the print function.

>>> Mariners\_runs = (10, 8, 8, 2, 3, 3, 5, 0, 6, 1);

>>> print(Mariners\_runs)

Note that parentheses are the syntax Python uses for the creation of a tuple, similar to the use of single of double quotes to create a string. Commas are required to separate the **elements** of the tuple.

**(10, 8, 8, 2, 3, 3, 5, 0, 6, 1)**

**b.** Create a tuple containing the runs scored by the Mariner’s opponents over the first 10 games, and name this list Opponents\_runs.

**2.** **Accessing elements of the tuple**

**a.** Enter the following commands and write down the results

>>> Mariners\_runs[6]

>>> Opponents\_runs[5]

>>> Mariners\_runs[1]

>>> Mariners\_runs[0]

>>> Mariners\_runs[10]

As you may suspect, the number inside the square brackets specifies the position of the **element** inthe list you want to **access.** The number specifying the position of an element is often called the element’s **index**. **The index of the first element of any sequence in Python is always 0.** This means that the second element in the list corresponds to an index of 1. This way of counting takes some getting used to...

**5**

**6**

**8**

**10**

**IndexError: tuple index out of range**

**b.** Write an expression that will retrieve the number of runs scored by the Mariners opponents in their 10th game.

**Opponents\_runs[9]**

**c.** Re-run the following command, then write down and explain the error message that results.

>>> Mariners\_runs[10]

**IndexError: tuple index out of range**

**This error occurs because Mariners\_runs[10] doesn’t exist, as only 10 games are recorded from index 0 to index 9.**

**d.** Write a program that looks at the tuples containing the Mariners runs and their opponents runs and determines the Mariner's overall record. You will need to “loop through” the tuples and compare the number of runs for each game. Use the runs data from this Lab.

**Mariners\_record = 0**

**for i in range(10):**

**if Mariners\_runs[i] > Opponents\_runs[i]:**

**Mariners\_record += 1**

**print(Mariners\_record)**

**3.** In Python, the elements of a tuple need not be of the same type. Run the following statements to see how this works.

>>> crazy\_tuple = (42, "hello", 3.14159, True)

>>> print(crazy\_tuple)

>>> crazy\_tuple[1]

**We will rarely if ever use this feature of tuples in Python.** In most programming languages, **arrays** (sequences like these) require all their elements to be of the same type – all ints, all floats, all Booleans, etc.

**4.** **A string is a sequence of characters.**

We already think of strings as variables that contain text. Since text is an ordered set of characters, it is reasonable that a string should be thought of as a sequence of characters.

**a. Strings can be indexed –**Run the following commands to see what this means.

>>> team\_name = 'Mariners'

>>> team\_name[0]

>>> team\_name[1]

>>> len(team\_name)

>>> team\_name[8]

What happened when you ran the last command and why? Don't forget to read the error messages to help figure out what's going on. This will be helpful when the errors are more complex.

**The final command tries to access a value that is out of range of the sequence of characters, so it throws an IndexError.**

**b.** **Strings (and tuples) are immutable –** You cannot change or delete an element of a string once it has been created. Try these commands and watch them fail...

>>> team\_name[5] = 'o'

>>> del team\_name[5]

**Both fail, proving strings are immutable.**

**5.** **More indexing**   
Do the following at the command line to see an interesting twist. Write down the results of each command.

>>> animals = ('cat', 'dog', 'bird', 'rabbit', 'lizard')

>>> animals[1]

>>> animals[3]

>>> animals[-1]

>>> animals[-3]

>>> animals[-len(animals)]

>>> animals[len(animals)]

>>> animals[3] = 'ferret'

**a.** Did animals[1]; return the first element of the tuple? Explain.

**It returned the second element of the tuple because indexing begins at 0.**

**b.** What happens when you use a negative number as an index?

**It goes to the end of the tuple and continues left. [-1] accesses the last element of the tuple.**

**c.** Why did the last two commands fail?

**Since the length of the array is one more than what its indices extend to, [len(ex\_tuple)] will always be out of range.**

**Tuples are immutable, so assignment to a specific index will throw an error.**

**6.** **Slicing**

The colon, :, allows you to access subsequences of a given sequence

>>> animals[1:3]

>>> animals[:3]

>>> animals[3:]

>>> animals[:]

>>> animals[-2:]

**a.** Write down and then test an expression that will pull out the 3rd and 4th elements of animals.

**animals[2:4]**

**b.** Write down and then test an expression that will pull out the 3rd and 4th elements of animals using only negative index numbers.

**animals[-3:-1]**

**c.** Run the following lines and then explain the 2nd line below:

>>> animals;

>>> new\_zoo = animals[:3] + ('ferret',) +animals[3:];

>>> new\_zoo;

Why is the comma needed after the string 'ferret'?

**Because without the comma, the parentheses are only interpreted as containing an expression, which in this case is a string, and a string cannot be concatenated with a tuple.**

d. Using the same method as in part c, add a line that will replace the word 'ferret' with the word 'meercat'. Even though tuples are immutable so you can't actually replace one element with another, this is one method to create a new tuple that is the same as the old just with one element different.

**even\_newer\_zoo = new\_zoo[:3] + ('meercat',) + new\_zoo[4:]**

**7.** **Nested Sequences**

The tuple is actually a nested sequence because the elements of the animals tuple, which is

a sequence, are themselves sequences (strings to be specific). Run the following commands

and write down the results.

>>> animals[2][0];

>>> animals[2][1];

>>> animals[2][2];

>>> animals[2][2];

>>> animals[2][:2];

>>> animals[-1][-4:];

**b**

**i**

**r**

**r**

**bi**

**zard**

1. Write down and then test an expression that will pull out the two letter b’s in the word rabbit.

**animals[3][2:4]**

1. Now write down the Mariners and their opponents scores in a single nested tuple called scores. A call of scores[0][0] should give the Mariner's score in the first game, and a call of scores[6][1] should give the opponents score in the 7th game.

**scores = (Mariners\_runs, ) + (Opponents\_runs, )**

**This works for the first one, but scores[6][1] is out of range. However, the presumably correctly placed scores[1][6] gives their opponent’s score in the 6th game.**