

Splitting and Joining Strings

Abdulaziz M. Alqumayzi

(DS-540) – Intro to python for computer science and data science: Learning to program with

AI, Big Data and The Cloud

Colorado State University – Global Campus

Dr. Ernest Bonat

October 24, 2020

Splitting and Joining Strings

Introduction

In this critical thinking activity, we will talk how splitting and joining strings in Python environment. The brain splits it into individual phrases, or tokens, each of which conveys significance, as you read a phrase. Interpreters such as IPython tokenize sentences, splitting them into individual elements of a programming language, such as keywords, identifiers, operators, and other elements. Tokens are usually divided by whitespace characters such as void, tab and newline, however the separators are regarded as delimiters, other characters may be included.

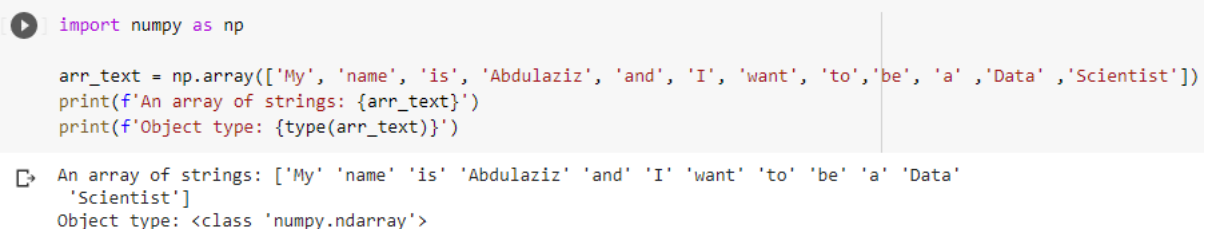
Discussion

Build a ten-string array.

Building a ten-string array. Actually, in this activity a twelve-string array was built. The array contains the following sentence “my name is Abdulaziz and I want to be a Data Scientist” divided into items as you will be shown in figure 1. The code started by importing NumPy library as np. Follow it the built array. Then, the array was printed and the type of this object, which is NumPy array denoted by `numpy.ndarray`.

Figure 1

Building a ten-string array code and output



```
import numpy as np

arr_text = np.array(['My', 'name', 'is', 'Abdulaziz', 'and', 'I', 'want', 'to', 'be', 'a', 'Data', 'Scientist'])
print(f'An array of strings: {arr_text}')
print(f'Object type: {type(arr_text)}')
```

➤ An array of strings: ['My' 'name' 'is' 'Abdulaziz' 'and' 'I' 'want' 'to' 'be' 'a' 'Data' 'Scientist']
Object type: <class 'numpy.ndarray'>

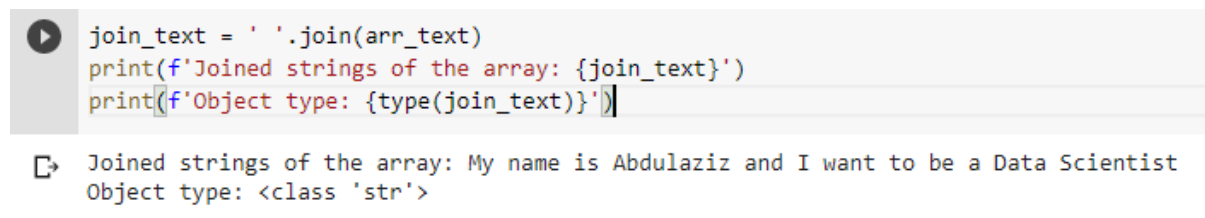
Note. In figure 1, the code is next to the circle that has a triangle inside it and the output is next to the rectangle with the arrow that coming from it.

Create a joining and splitting program for these strings.

Python `join()` function is used to concatenate the strings in the array. This function concatenate all strings in the array and separated it by a space ' '.join as it shown in figure 2. The output printed after the join and we can see that it became a full sentence separated by spaces. The type of the `join_text` object is `str`, which means a string type. The `join()` function converts the object type from `ndarray` into `str`.

Figure 2

joining strings from an array code and output



```
join_text = ' '.join(arr_text)
print(f'Joined strings of the array: {join_text}')
print(f'Object type: {type(join_text)}')
```

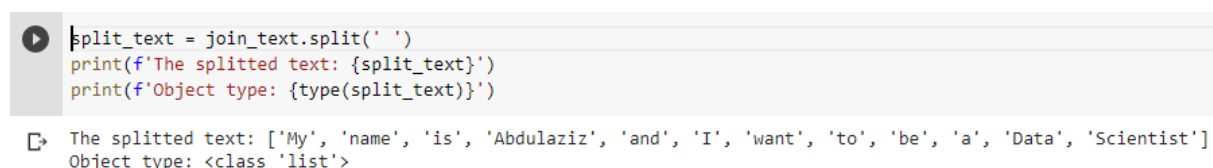
Joined strings of the array: My name is Abdulaziz and I want to be a Data Scientist
Object type: <class 'str'>

Note. In figure 2, the code shows a join function to an array and two printed output, one for the `join_text` object and one for the type of the `join_text` object.

Python `split()` function used to substrings the sentence into a list of strings. In figure 3, the sentence was splatted by the space between each word and stored in `split_text` object. The `split_text` object type converted from `str` (String) into `list`. Always the `split()` function converts from a `str` object into a `list` object.

Figure 3

splitting sentence into a list codes and outputs



```
split_text = join_text.split(' ')
print(f'The splitted text: {split_text}')
print(f'Object type: {type(split_text)}')
```

The splitted text: ['My', 'name', 'is', 'Abdulaziz', 'and', 'I', 'want', 'to', 'be', 'a', 'Data', 'Scientist']
Object type: <class 'list'>

Note. In figure 3, the code shows in the first line how to use `split()` function to split a string object. The following codes are to print the list object `split_text` and the type of the object which is a list.

Other data types joining and splitting.

We talked about joining and splitting Python array data type. But what about the other data types. Tuple data type is typical to the array data type in joining the strings that tuple has. In figure 4, there are two snippets of codes, the first snippet shows the creation of a tuple as `tuple_text` and it printed with its data type to show its object type. Then the second snippet shows the joining of the elements in the tuple as it typical from the code shown in figure 2. The joining of the tuple converted the tuple into str object, which is mean the splitting will be the same to the previous code in figure 3.

Figure 4

build and joining a tuple code and output

```
[146] tuple_text = ('This', 'is', 'the', 'tuple', 'example')
      print(f'The set of strings: {tuple_text}')
      print(f'Object type: {type(tuple_text)}')
```

```
The set of strings: ('This', 'is', 'the', 'tuple', 'example')
Object type: <class 'tuple'>
```

```
[147] tuple_join = ' '.join(tuple_text)
      print(f'Joined strings of the dictionary: {tuple_join}')
      print(f'Object type: {type(tuple_join)}')
```

```
Joined strings of the dictionary: This is the tuple example
Object type: <class 'str'>
```

Note. In figure 4, two snippet codes to show building and joining elements of a tuple data type.

The second data type that we will talk about is the dictionary data type. Joining dictionary data type needs more effort, due to the structure of the dictionary is vary a lot from

tuple, list and array which we can access their elements by indexes. In the dictionary we need to extract the values, not keys of the values, So for loop was used to extract these values as shown in figure 5. In figure 5, first snippet shows the creation of the dictionary and second snippet shows the for loop for extracting the values that we need. Again, join() function converted the object data type into str, that means splitting the string will be the same step in figure 3.

Figure 5

build and joining a dictionary code and output

```
[148] dict_text = {1:'This', 2:'is', 3:'the', 4:'dictionary', 5:'example'}
      print(f'The dictionary of strings: {dict_text}')
      print(f'Object type: {type(dict_text)}')
```

The dictionary of strings: {1: 'This', 2: 'is', 3: 'the', 4: 'dictionary', 5: 'example'}
Object type: <class 'dict'>

```
[149] dict_join = ' '.join(i for i in dict_text.values())
      print(f'Joined strings of the dictionary: {dict_join}')
      print(f'Object type: {type(dict_join)}')
```

Joined strings of the dictionary: This is the dictionary example
Object type: <class 'str'>

Note. In figure 5, two snippet codes to show building and joining elements of a dictionary data type.

Lastly, the set data type; due to its unordered elements nature. It is difficult to handle and manipulate a set object. In figure 6, the first snippet shows the creation of the set object data type set_text. We can see from the printed code the arrange of the elements different from when we store it. The second snippet shows the output sentence from the set_join object is varying in arrangement from when we store that elements in set_text object. Therefore, we must be careful when we want to store string elements in a set data type object.

Figure 6

build and joining a set code and output

```
▶ set_text = {'This', 'is', 'the', 'set', 'example'}  
print(f'The set of strings: {set_text}')  
print(f'Object type: {type(set_text)}')
```

↪ The set of strings: {'example', 'set', 'is', 'the', 'This'}
Object type: <class 'set'>

```
[164] set_join = ' '.join(set_text)  
print(f'Joined strings of the dictionary: {set_join}')  
print(f'Object type: {type(set_join)}')
```

Joined strings of the dictionary: example set is the This
Object type: <class 'str'>

Note. In figure 6, two snippet codes to show building and joining elements of a set data type.

References

Deitel, P. J., & Deitel, H. M. (2020). *Intro to Python for computer Science and data science learning to program with AI, big data and the cloud*. Hudson Street, NY: Pearson Education.

Kalb, I. (2016). *Learn to program with Python*. Berkeley, CA: Apress.

Mueller, J., & Emid, A. (2018). *Beginning programming with Python® for dummies®*. Hoboken, NJ: For Dummies, a Wiley brand.