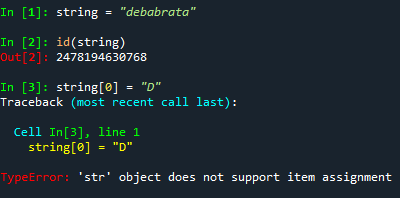
Q1. Does assigning a value to a string's indexed character violate Python's string immutability?

Answer:

Since strings are immutable, the indexed character cannot be assigned a new value.

Example:

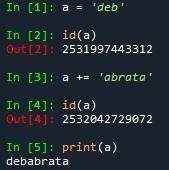


Q2. Does using the += operator to concatenate strings violate Python's string immutability? Why or why not?

Answer:

The += operator does not violate Python’s string immutability while concatenate strings since we are not altering string based on the indexes instead we are increasing or adding new indexes while concatenating strings. Also, it does not modify existing string instead it creates a new string bbject.

Example:



Q3. In Python, how many different ways are there to index a character?

Answer:

Characters within a string can be indexed using the character index number in square brackets ([]) after the string name. Possible ways of index a character:

🡪 Positive index where the first index is 0 and so on

🡪 Negative index where the last character is -1 and so on

Example:

name = "debabrata"

print(name[0], name[5], name[9])

print(name[-1], name[-5], name[-8])

Q4. What is the relationship between indexing and slicing?

Answer:

Used to access the elements of a sequence data type using slicing and indexing. Index is used to get a single element while slicing a set of elements.

Example:

a\_string = "debabrata"

print(a\_string[1]) # Indexing

print(a\_string[0:4]) # Slicing

Q5. What is an indexed character's exact data type? What is the data form of a slicing-generated substring?

Answer:

An indexed character's exact data type is "str" type and the data form of a slicing-generated is also "str" type.

Example:

a = "debabrata"

print(type(a[2])) # Indexing -> str

print(type(a[4:8])) # Slicing -> str

Q6. What is the relationship between string and character "types" in Python?

Answer:

An object containing a sequence of character data types is called a string. A Python string is a sequence of characters. Each character in a string is represented by a Unicode code point. A character, on the other hand, refers to a single Unicode code point.

Q7. Identify at least two operators and one method that allow you to combine one or more smaller strings to create a larger string.

Answer:

🡪 + (or, +=) and \* are allow to combine one or more smaller strings to create a larger string.

🡪 <string>.join(<sep>) method joins element of iterable type like list and tuple to get a combined string.

Example:

string = "Hello," + "World! "

string += "Hello, World!"

print(string)

s = " ".join(['Hello,', 'World!'])

print(s)

s = " ".join(('Hello,', 'World!'))

print(s)

Q8. What is the benefit of first checking the target string with in or not in before using the index method to find a substring?

Answer:

Checking the target string using the in or not operator before searching for a substring using the index method is only useful for checking the availability of the substring, hence the ValueError avoid occurrence.

Example:

s = 'abcde'

s.index('g') # raises ValueError

s.index('e') # it will return 4 as an output

Q9. Which operators and built-in string methods produce simple Boolean (true/false) results?

Answer:

The operators and built-in string methods produce simple Boolean (true/false) results are following:

* Operators:
  + ==, !=
  + in, not in
* built-in methods:
  + <string>.startswith()
  + <string>.endswith()
  + <string>.isalpha()
  + <string>.isalnum()
  + <string>.isdecimal()
  + <string>.isdigit()
  + <string>.isidentifier()
  + <string>.islower()
  + <string>.isnumeric()
  + <string>.isprintable()
  + <string>.isspace()
  + <string>.istitle()
  + <string>.isupper()