Strings Revisited

We have used strings to represent words or phrases that we wanted to print out. Our definition was simple: a string is simply some characters inside quotes. In this chapter we explore strings in much more detail.

A Collection Data Type

So far we have seen built-in types like: `int` , `float` , `bool` , `str` and we've seen lists. `int` , `float` , and `bool` are considered to be simple or primitive data types because their values are not composed of any smaller parts. They cannot be broken down. On the other hand, strings and lists are different from the others because they are made up of smaller pieces. In the case of strings, they are made up of smaller strings each containing one **character**.

Types that are comprised of smaller pieces are called **collection data types**. Depending on what we are doing, we may want to treat a collection data type as a single entity (the whole), or we may want to access its parts. This ambiguity is useful.

Strings can be defined as sequential collections of characters. This means that the individual characters that make up the string are assumed to be in a particular order from left to right.

A string that contains no characters, often referred to as the **empty string**, is still considered to be a string. It is simply a sequence of zero characters and is represented by '' or "" (two single or two double quotes with nothing in between).

Operations on Strings

In general, you cannot perform mathematical operations on strings, even if the strings look like numbers. The following are illegal (assuming that `message` has type string):

```
message - 1
"Hello" / 123
message * "Hello"
"15" + 2
```

Interestingly, the `+` operator does work with strings, but for strings, the `+` operator represents **concatenation**, not addition. Concatenation means joining the two operands by linking them end-to-end. For example:

```
1 fruit = "banana"
2 bakedGood = " nut bread"
3 print(fruit + bakedGood)
4
```

ActiveCode: 1 (ch08_add)

Run

```
banana nut bread
```

The output of this program is `banana nut bread`. The space before the word `nut` is part of the string and is necessary to produce the space between the concatenated strings. Take out the space and run it again.

The `*` operator also works on strings. It performs repetition. For example, `'Fun'*3` is `'FunFunFun'`. One of the operands has to be a string and the other has to be an integer.

```
print("Go"*6)

name = "Packers"
print(name * 3)

print(name + "Go" * 3)

print((name + "Go") * 3)

```

ActiveCode: 2 (ch08_mult)

Run

```
GoGoGoGoGoGo
PackersPackersPackers
PackersGoGoGo
PackersGoPackersGoPackersGo
```

This interpretation of `+` and `*` makes sense by analogy with addition and multiplication. Just as `4*3` is equivalent to `4+4+4`, we expect `"Go"*3` to be the same as `"Go"+"Go"+"Go"`, and it is. Note also in the last example that the order of operations for `*` and `+` is the same as it was for arithmetic. The repetition is done before the concatenation. If you want to cause the concatenation to be done first, you will need to use parenthesis.

Check your understanding

str-1: What is printed by the following statements?

```
s = "python"
t = "rocks"
print(s+t)
```

- a) python rocks
- b) python
- c) pythonrocks
- d) Error, you cannot add two strings together.

Check Me | Compare Me

Correct!! Yes, the two strings are glued end to end.

str-2: What is printed by the following statements?

```
s = "python"
excl = "!"
print(s+excl*3)
```

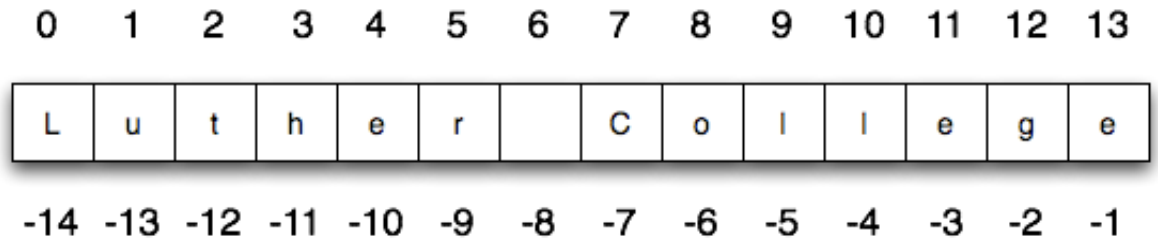
- a) python!!!
- b) python!python!python!
- c) pythonpythonpython!
- d) Error, you cannot perform concatenation and repetition at the same time.

Check Me | Compare Me

Correct!! Yes, repetition has precedence over concatenation

Index Operator: Working with the Characters of a String

The **indexing operator** (Python uses square brackets to enclose the index) selects a single character from a string. The characters are accessed by their position or index value. For example, in the string shown below, the 14 characters are indexed left to right from postion 0 to position 13.



It is also the case that the positions are named from right to left using negative numbers where -1 is the rightmost index and so on. Note that the character at index 6 (or -8) is the blank character.

```
1 school = "Luther College"
2 m = school[2]
3 print(m)
4
5 lastchar = school[-1]
6 print(lastchar)
7
```

ActiveCode: 3 (chp08_index1)

Run

```
t
e
```

The expression `school[2]` selects the character at index 2 from `school` , and creates a new string containing just this one character. The variable `m` refers to the result.

Remember that computer scientists often start counting from zero. The letter at index zero of `"Luther College"` is `L` . So at position `[2]` we have the letter `t` .

If you want the zero-eth letter of a string, you just put 0, or any expression with the value 0, in the brackets. Give it a try.

The expression in brackets is called an **index**. An index specifies a member of an ordered collection. In this case the collection of characters in the string. The index *indicates* which character you want. It can be any integer expression so long as it evaluates to a valid index value.

Note that indexing returns a *string* — Python has no special type for a single character. It is just a string of length 1.

Check your understanding

str-3: What is printed by the following statements?

```
s = "python rocks"
print(s[3])
```

- a) t
- b) h
- c) c
- d) Error, you cannot use the [ ] operator with a string.

Check Me | Compare Me

Correct!! Yes, index locations start with 0.

str-4: What is printed by the following statements?

```
s = "python rocks"
print(s[2] + s[-5])
```

- a) tr
- b) ps
- c) nn
- d) Error, you cannot use the [ ] operator with the + operator.

Check Me | Compare Me

Correct!! Yes, indexing operator has precedence over concatenation.

String Methods

We previously saw that each turtle instance has its own attributes and a number of methods that can be applied to the instance. For example, we wrote `tess.right(90)` when we wanted the turtle object `tess` to perform the `right` method to turn to the right 90 degrees. The “dot notation” is the way we connect the name of an object to the name of a method it can perform.

Strings are also objects. Each string instance has its own attributes and methods. The most important attribute of the string is the collection of characters. There are a wide variety of methods. Try the following program.

```
ss = "Hello, World"
print(ss.upper())

tt = ss.lower()
print(tt)

```

ActiveCode: 4 (chp08_upper)

Run

```
HELLO, WORLD
hello, world
```

In this example, `upper` is a method that can be invoked on any string object to create a new string in which all the characters are in uppercase. `lower` works in a similar fashion changing all characters in the string to lowercase. (The original string `ss` remains unchanged. A new string `tt` is created.)

In addition to `upper` and `lower`, the following table provides a summary of some other useful string methods. There are a few activecode examples that follow so that you can try them out.

Method	Parameters	Description
upper	none	Returns a string in all uppercase
lower	none	Returns a string in all lowercase
capitalize	none	Returns a string with first character capitalized, the rest lower
strip	none	Returns a string with the leading and trailing whitespace removed
lstrip	none	Returns a string with the leading whitespace removed
rstrip	none	Returns a string with the trailing whitespace removed

count	item	Returns the number of occurrences of item
replace	old, new	Replaces all occurrences of old substring with new
center	width	Returns a string centered in a field of width spaces
ljust	width	Returns a string left justified in a field of width spaces
rjust	width	Returns a string right justified in a field of width spaces
find	item	Returns the leftmost index where the substring item is found
rfind	item	Returns the rightmost index where the substring item is found
index	item	Like find except causes a runtime error if item is not found
rindex	item	Like rfind except causes a runtime error if item is not found

You should experiment with these methods so that you understand what they do. Note once again that the methods that return strings do not change the original. You can also consult the Python documentation for strings (http://docs.python.org/py3k/library/stdtypes.html#index-21).

```
ss = "    Hello, World    "

els = ss.count("l")
print(els)

print("***"+ss.strip()+"***")
print("***"+ss.lstrip()+"***")
print("***"+ss.rstrip()+"***")

news = ss.replace("o", "***")
print(news)

```

ActiveCode: 5 (ch08_methods1)

Run

```
3
***Hello, World***
***Hello, World    ***
***    Hello, World***
    Hell***, W***rld
```

```
1  food = "banana bread"
2  print(food.capitalize())
3
4  print("*"+food.center(25)+"*")
5  print("*"+food.ljust(25)+"*")      #stars added to show bounds
6  print("*" +food.rjust(25)+"*")
7
8  print(food.find("e"))
9  print(food.find("na"))
10 print(food.find("b"))
11
12 print(food.rfind("e"))
13 print(food.rfind("na"))
14 print(food.rfind("b"))
15
```

ActiveCode: 6 (ch08_methods2)

Run

```
Banana bread
*      banana bread       *
*banana bread             *
*             banana bread*
9
2
0
9
4
7
9
```

Check your understanding

str-5: What is printed by the following statements?

```
s = "python rocks"
print(s.count("o") + s.count("p"))
```

- a) 0
- b) 2
- c) 3

Check Me | Compare Me

Correct!! Yes, add the number of o characters and the number of p characters.

str-6: What is printed by the following statements?

```
s = "python rocks"
print(s[1]*s.index("n"))
```

- a) yyyyy
- b) 55555
- c) n
- d) Error, you cannot combine all those things together.

Check Me | Compare Me

Correct!! Yes, s[1] is y and the index of n is 5, so 5 y characters. It is important to realize that the index method has precedence over the repetition operator. Repetition is done last.

Length

The `len` function, when applied to a string, returns the number of characters in a string.

```
1 fruit = "Banana"
2 print(len(fruit))
3
```

ActiveCode: 7 (chp08_len1)

Run

```
6
```

To get the last letter of a string, you might be tempted to try something like this:

```
1 fruit = "Banana"
2 sz = len(fruit)
3 last = fruit[sz]        # ERROR!
4 print(last)
5
```

ActiveCode: 8 (chp08_len2)

Run

That won't work. It causes the runtime error `IndexError: string index out of range`. The reason is that there is no letter at index position 6 in `"Banana"`. Since we started counting at zero, the six indexes are numbered 0 to 5. To get the last character, we have to subtract 1 from `length`. Give it a try in the example above.

```
1 fruit = "Banana"
2 sz = len(fruit)
3 lastch = fruit[sz-1]
4 print(lastch)
5
```

ActiveCode: 9 (ch08_len3)

Run

```
a
```

Typically, a Python programmer will access the last character by combining the two lines of code from above.

```
lastch = fruit[len(fruit)-1]
```

Check your understanding

str-7: What is printed by the following statements?

```
s = "python rocks"
print(len(s))
```

- a) 11
- b) 12

Check Me | Compare Me

Correct!! Yes, there are 12 characters in the string.

str-8: What is printed by the following statements?

```
s = "python rocks"
print(s[len(s)-5])
```

- a) o
- b) r
- c) s
- d) Error, len(s) is 12 and there is no index 12.

Check Me | Compare Me

Correct!! Yes, len(s) is 12 and 12-5 is 7. Use 7 as index and remember to start counting with 0.

The Slice Operator

A substring of a string is called a **slice**. Selecting a slice is similar to selecting a character:

```
1 singers = "Peter, Paul, and Mary"
2 print(singers[0:5])
3 print(singers[7:11])
4 print(singers[17:21])
5
```

ActiveCode: 10 (chp08_slice1)

Run

```
Peter
Paul
Mary
```

The slice operator `[n:m]` returns the part of the string from the n'th character to the m'th character, including the first but excluding the last. In other words, start with the character at index n and go up to but do not include the character at index m. This behavior may seem counter-intuitive but if you recall the `range` function, it did not include its end point either.

If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string.

```
1 fruit = "banana"
2 print(fruit[:3])
3 print(fruit[3:])
4
```

ActiveCode: 11 (chp08_slice2)

Run

```
ban
ana
```

What do you think `fruit[:]` means?

Check your understanding

str-9: What is printed by the following statements?

```
s = "python rocks"
print(s[3:8])
```

- a) python
- b) rocks
- c) hon r
- d) Error, you cannot have two numbers inside the [ ].

Check Me | Compare Me

Correct!! Yes, start with the character at index 3 and go up to but not include the character at index 8.

str-10: What is printed by the following statements?

```
s = "python rocks"
print(s[7:11]*3)
```

- a) rockrockrock
- b) rock rock rock
- c) rocksrocksrocks
- d) Error, you cannot use repetition with slicing.

Check Me | Compare Me

Correct!! Yes, rock starts at 7 and goes thru 10. Repeat it 3 times.

Note

This workspace is provided for your convenience. You can use this activecode window to try out anything you like.

```
1
2
```

ActiveCode: 12 (scratch_08_01)

Run

Error

```
IndexError: string index out of range on line 3
```

Description

This message means that you are trying to index past the end of a string or a list. For example if your list has 3 things in it and you try to access the item at position 3 or more.

To Fix

Remember that the first item in a list or string is at index position 0, quite often this message comes about because you are off by one. Remember in a list of length 3 the last legal index is 2