COSC2429 Introduction to Programming

Strings

Quang Tran <quang.tran@rmit.edu.vn>

RMIT UNIVERSITY

Outline

- String revisited
- + and * operator
- Index operator
- String methods
- Summary of string methods
- String length
- Slice operator
- String comparison
- Strings are immutable
- Traversal by character
- Traversal by index
- in and not in operator
- Example: Removing vowels in a string
- Example: Counting a character in a string
- Example: Finding the index of a character in a string
- Character classification

String revisited

- So far we have learned 4 basic built-in data types: **int**, **float**, **bool**, **str**.
- A string is a **sequence of characters**.
- A string that contains no characters, often referred to as the **empty string**, is still considered to be a string.

```
s1 = "Hello, World"
s2 = ""
print(s1)
print(s2)
```

+ and * operator

- The + operator works with strings. It represents **concatenation**, which means joining the operands by linking them together.

```
fruit = "banana"
baked_good = " nut bread”
print(fruit + baked_good)
```

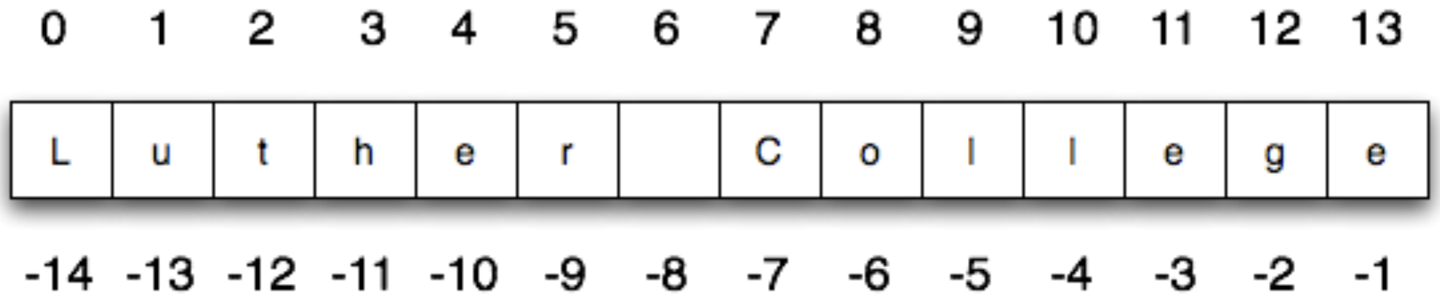
- The * operator also works on strings. It performs repetition on a given string. Thus, one of the operands has to be a string and the other has to be an integer.
- Other mathematical operators don’t work with strings.

```
print("Go" * 6)

name = "RMIT"
print(name * 3)
print(name + "Go" * 3)
print((name + "Go") * 3)
```

Index operator

- The **index 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**.
- The indexes are named from left to right using positive numbers or from right to left using negative numbers.



```
school = "Luther College"
m = school[2]
print(m)

last_char = school[-1]
print(last_char)
```

String methods

- Like turtles, strings are objects. Each object has its own attributes and methods. To call a method, we write **object.method()**. For example:

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

tt = ss.lower()
print(tt)
```

- Another example:

```
ss = "    Hello, World    "

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

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

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

String methods

- And another example:

```
food = "banana bread"
print(food.capitalize())

print("*" + food.center(25) + "*")
print("*" + food.ljust(25) + "*")          # stars added to show bounds
print("*" + food.rjust(25) + "*")

print(food.find("e"))
print(food.find("na"))
print(food.find("b"))

print(food.rfind("e"))
print(food.rfind("na"))
print(food.rfind("b"))
```

*** Summary of string methods ***

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

String length

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

```
fruit = "Banana”
print(len(fruit))
```

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

```
fruit = "Banana"
sz = len(fruit)
last = fruit[sz]       # ERROR!
print(last)
```

- That won’t work. It causes the runtime error **IndexError: string index out of range**. Do you know the reason? The error is fixed as follow:

```
fruit = "Banana"
sz = len(fruit)
last = fruit[sz - 1]
print(last)
```

Slice operator

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

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

- The slice operator **[start:stop]** returns the substring from the character at index **start** and go up to but do not include the character at index **stop**. This behavior is similar to the behavior of the function **range** that we studied earlier.
- 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.

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

String comparison

- To see if two strings are equal you simply write a boolean expression using the equality operator:

```
word = "banana"
if word == "banana":
    print("Yes, we have bananas!")
else:
    print("Yes, we have NO bananas!")
```

- Other comparison operations are useful for putting words in lexicographical order. This is similar to the alphabetical order in a dictionary, except that all the uppercase letters come before all the lowercase letters.

```
word = "zebra"

if word < "banana":
    print("Your word, " + word + ", comes before banana.")
elif word > "banana":
    print("Your word, " + word + ", comes after banana.")
else:
    print("Yes, we have no bananas!")
```

String comparison

- But what if we consider the words “apple” and “Apple”? Are they the same?

```
print("apple" < "banana")
print("apple" == "Apple")
print("apple" < "Apple")
```

- It turns out that uppercase and lowercase letters are considered to be different from one another.
- In computing, each character is assigned a unique integer value. For example, “A” is 65, “B” is 66, and “5” is 53.
- The way you can find out the so called **ordinal value** for a given character is to use a character function called **ord**.

```
print(ord("A"))
print(ord("B"))
print(ord("5"))

print(ord("a"))
print("apple" > "Apple")
```

String comparison

- There is also a similar function called **chr** that converts integers into their character equivalent.

```
print(chr(65))
print(chr(66))

print(chr(49))
print(chr(53))

print("The character for 32 is", chr(32), "!!!")
print(ord(" "))
```

Strings are immutable

- Strings are **immutable**, which means you cannot change an existing string.

```
greeting = "Hello, world!"
greeting[0] = 'J'            # ERROR!
print(greeting)
```

- The best you can do is create a new string that is a variation on the original.

```
greeting = "Hello, world!"
new_greeting = 'J' + greeting[1:]
print(new_greeting)
print(greeting)            # same as it was
```

Traversal by character

- A lot of computations involve processing a collection one item at a time. Recall the example of a for loop with a list of items:

```
for name in ["Joe", "Amy", "Brad", "Angelina", "Zuki", "Thandi", "Paris"]:
    invitation = "Hi " + name + ". Please come to my party on Saturday!"
    print(invitation)
```

- Or another example where the sequence of integers created by the range function:

```
for value in range(10):
    print(value)
```

- A string is a sequence of characters thus we can also use it in a for loop:

```
for char in "Go Spot Go":
    print(char)
```

Traversal by index

- We can also use the for loop with the indexing operator to access the individual characters in the string as follow.

```
fruit = "apple"
for index in range(len(fruit)):
    print(fruit[index])
```

- The above code print the characters from left to right. What if we want to print the characters from right to left?
- No problem, we can create a range that counts down as follow:

```
fruit = "apple"
for index in range(len(fruit)-1, -1, -1):
    print(fruit[index])
```

in and **not in** operators

- The **in** operator tests if one string is a substring of another:

```
print('p' in 'apple')
print('i' in 'apple')
print('ap' in 'apple')
print('pa' in 'apple')
print('' in 'apple')
```

- The **not in** operator returns the logical opposite result of in.
- How's about the empty string? Is it a substring of another string?

```
print('x' not in 'apple')
print('' not in 'apple')
```

Example: Removing vowels from a string

- Combining the in and + operator with string, we can write a function that removes all the vowels from a string as follow:

```
# Function definition
def remove_vowels(s):
    """
    Return a new string created by removing all vowels from string s
    :param s: a string
    :return: a new string created by removing all vowels from s
    """
    vowels = "aeiouAEIOU"
    new_str = ""
    for char in s:                          # process each character of s
        if char not in vowels:
            new_str = new_str + char        # only add char if it's not a vowel
    return new_str

# Main program to test the function
print(remove_vowels("compsci"))
print(remove_vowels("aAbEefIijOopUus"))
```

Example: Counting a letter in a string

- Here is a function that counts the number of times a given letter appearing in a string:

```
# Function definition
def count(text, letter):
    """
    Return the number of the given letter in text
    :param text: a string
    :param letter: a letter
    :return: number of occurrences of letter in text
    """
    cnt = 0
    for char in text:
        if char == letter:
            cnt = cnt + 1
    return cnt

# Main program to test the function
print(count("banana", "a"))
```

Example: Finding index of a character in a string

```
# Function definition
def find(s, char):
    """
    Find the index of the character char in string s
    :param s: a string
    :param char: a character
    :return:  the index of char in s or -1 if char is not in s
    """
    index = 0
    while index < len(string):
        if string[index] == char:
            return index # char in s
        else:
            index = index + 1

    return -1               # char not found in s

# Main program to test the function
print(find("Compsci", "p"))
print(find("Compsci", "x"))
```

Character classification

- It is often helpful to examine a character and test whether it is upper- or lowercase, or whether it is a character or a digit. (which operator to use?)
- The **string** module provides several constants that are useful for these purposes.
- Here are some of those constants:

```
import string

print(string.ascii_lowercase)
print(string.ascii_uppercase)
print(string.digits)
print(string.punctuation)
```