05.1_Strings

September 18, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Strings:
  - slicing
  - comparison
  - methods

2 Standard Data Types in Python - strings

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	“Hello World”
	List	[1, “Hello World”]
	Tuple	(1, “Hello World”)
	Set	{1, “Hello World”}
	Dictionary	{1: “Hello World”, 2: 100}

3 Iterating over a string with `for` statements (for Loops) (traversal)

Traversal: start at the beginning, select each character in turn, do something to it, and continue until the end.

- `for` statments are used to iterate over sequences
- `for/range` statments are used to iterate over sequences using an index

[1]:
```
a = "python is fun!"
```

[2]:
```
"is" in a
```

[2]: True

[3]:
```
for i in a:
    print(i)
```

```
p
y
t
h
o
n

i
s

f
u
n
!
```

[4]:
```
len(a)
```

[4]: 14

[5]:
```
for i in range(len(a)):
    print(i, a[i])
```

```
0 p
1 y
2 t
3 h
4 o
5 n
6
7 i
```

```
8 s
9
10 f
11 u
12 n
13 !
```

3.1 Slicing strings

To access a continuous segment in a string

Structure of slicing a string: `string[start_index:end_index]` * string name * square brackets * `start`: the index to begin the slice * Colon `:` * `end`: the (non-inclusive) index to finish the slice

[6]:
```
my_string = "Hello World"
```

[7]:
```
my_string[0:5]
```

[7]:
```
'Hello'
```

Slice from the beginning of the string: `start` can be ignored

[8]:
```
my_string[:5]
```

[8]:
```
'Hello'
```

Slice all the way to the end of the string: `end` can be ignored

[9]:
```
my_string
```

[9]:
```
'Hello World'
```

[10]:
```
my_string[6:]
```

[10]:
```
'World'
```

[11]:
```
my_string[6:len(my_string)]
```

[11]:
```
'World'
```

[12]:
```
my_string[len(my_string)]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[12], line 1
----> 1 my_string[len(my_string)]

IndexError: string index out of range
```

[13]:
```
my_string
```

[13]:
```
'Hello World'
```

[14]:
```
my_string[6:6]
```

[14]:
```
''
```

[15]:
```
my_string[6:7]
```

[15]:
```
'W'
```

[16]:
```
my_string[:]
```

[16]:
```
'Hello World'
```

[17]:
```
my_string[1.0:3.0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 1
----> 1 my_string[1.0:3.0]

TypeError: slice indices must be integers or None or have an __index__ method
```

3.1.1 Group exercise:

Write python code to grab the "gin" slice from the string eng_string= 'engineer'

```
eng_string = 'engineer'
```

When you are done, raise your hand!

[18]:
```
eng_string = 'engineer'
```

[19]:
```
eng_string[2:5]
```

[19]:
```
'gin'
```

[20]:
```
eng_string[-6:-3]
```

[20]:
```
'gin'
```

3.1.2 Group Exercises

1. Define a string variable for 'banana' and print out the first and last 'a'.
2. Using the same string, grab the 2 possible slices that correspond to the word 'ana' and print them out.

When you are done, raise your hand!

[21]:
```
b_string = "banana"
```

[22]:
```
b_string[1:4]
```

[22]:
```
'ana'
```

[23]:
```
b_string[3:6]
```

[23]:
```
'ana'
```

[24]:
```
b_string[3:]
```

[24]:
```
'ana'
```

[25]:
```
b_string[-3:]
```

[25]:
```
'ana'
```

[26]:
```
b_string[1]
```

[26]:
```
'a'
```

[27]:
```
b_string[5]
```

[27]:
```
'a'
```

[28]:
```
b_string[-1]
```

[28]:
```
'a'
```

3.1.3 `for` or `for/range` statement with string slice

Obtain each character of a substring

[29]:
```
fruit_string = "banana is very sweet"
```

[30]:
```
fruit_string[-5:]
```

[30]:
```
'sweet'
```

[31]:
```
for s in fruit_string[-5:]:
    print(s)
```

```
s
w
e
```

```
e
t
```

[32]:
```
len(fruit_string)
```

[32]:
```
20
```

[33]:
```
fruit_string[15:20]
```

[33]:
```
'sweet'
```

[34]:
```
list(range(15, 20))
```

[34]:
```
[15, 16, 17, 18, 19]
```

[35]:
```
for i in range(15, 20):
    print(fruit_string[i])
```

```
s
w
e
e
t
```

[36]:
```
for i in range(6):
    print(fruit_string[i])
```

```
b
a
n
a
n
a
```

[37]:
```
for i in fruit_string[:6]:
    print(i)
```

```
b
a
n
a
n
a
```

[38]:
```
fruit_string[-5:]
```

[38]:
```
'sweet'
```

[39]:
```
for i in fruit_string[-5:]:
    print(i)
```

```
s
w
e
e
t
```

[40]:
```
fruit_string
```

[40]: 'banana is very sweet'

[41]:
```
fruit_string[7:9]
```

[41]: 'is'

[42]:
```
for i in fruit_string[7:9]:
    print(i)
```

```
i
s
```

[43]:
```
for i in range(7,9):
    print(fruit_string[i])
```

```
i
s
```

3.1.4 *Translate that!*

What is indexing string in python? How do we implement it?

[44]:
```
a = "python"
```

[45]:
```
a[0]
```

[45]: 'p'

3.1.5 *Translate that!*

What is slicing string in python? How do we implement it?

[46]:
```
a
```

[46]: 'python'

[47]:
```
a[1:4]
```

[47]: 'yth'

[48]:
```
a[0:1]
```

[48]: 'p'

3.1.6 String comparison

Comparison operators `>`, `<`, `==`

[49]:
```
"banana" == "banana"
```

[49]: True

[50]:
```
"banana" > "pear"
```

[50]: False

[51]:
```
"banana" > "Pear"
```

[51]: True

[52]:
```
"Pear".lower()
```

[52]: 'pear'

[53]:
```
"banana" > "Pear".lower()
```

[53]: False

3.2 Built-in methods with strings

What is a method?

- functions associated with a particular data type or a class of objects (e.g., strings)
  - methods are essentially functions
- format: `mystring.method()`
- call a method: the dot operator
  - the method comes after the dot
  - the name of the particular object it acts on comes first

[54]:
```
AE_quote = "Everybody is a genius."
AE_quote
```

[54]: 'Everybody is a genius.'

[55]:
```
AE_quote.upper()
```

[55]: 'EVERYBODY IS A GENIUS.'

[56]:
```
AE_quote
```

[56]: 'Everybody is a genius.'

[57]:
```
AE_quote.lower()
```

[57]: 'everybody is a genius.'

[58]:
```
AE_quote
```

[58]: 'Everybody is a genius.'

[59]:
```
AE_quote.capitalize()
```

[59]: 'Everybody is a genius.'

[60]:
```
a = AE_quote.lower()
print(a)
print(a.capitalize())
```

```
everybody is a genius.
Everybody is a genius.
```

[61]:
```
AE_quote
```

[61]: 'Everybody is a genius.'

[62]:
```
AE_quote.lower().capitalize()
```

[62]: 'Everybody is a genius.'

3.2.1 `count()` method

- gives the number of ocurrences of a substring in a range
- *Syntax:*

```
str.count(substring, start, end)
```

- `start` and `end`
  - integers that indicate the indices where to start and end the count
  - optional, if omitted, the whole string is inspected
  - `end`: non-inclusive

[63]:
```
AE_quote
```

[63]: 'Everybody is a genius.'

[64]:
```
AE_quote.count("e")
```

[64]:
```
2
```

[65]:
```
AE_quote.count("e", 0, 10)
```

[65]:
```
1
```

[66]:
```
AE_quote.count('e', 0, len(AE_quote))
```

[66]:
```
2
```

[67]:
```
AE_quote
```

[67]:
```
'Everybody is a genius.'
```

[68]:
```
AE_quote[10]
```

[68]:
```
'i'
```

[69]:
```
AE_quote.count('e', 10, 20)
```

[69]:
```
1
```

[70]:
```
AE_quote.count('e', 0, 11)
```

[70]:
```
1
```

[71]:
```
AE_quote
```

[71]:
```
'Everybody is a genius.'
```

[72]:
```
AE_quote.count('Everybody')
```

[72]:
```
1
```

[73]:
```
AE_quote.count('EverybodyHello')
```

[73]:
```
0
```

3.2.2 `find()` method

- tells us if a string `'substr'` occurs in the string and return the index where the substring starts, otherwise it will return `-1`.
- *Syntax:*

```
str.find(substring, start, end)
```

- `start` and `end`

- integers that indicate the indices where to start and end the count
- optional, if omitted, the whole string is inspected

[74]:
```
AE_quote = "Everybody is a genius."
AE_quote
```

[74]:
```
'Everybody is a genius.'
```

[75]:
```
AE_quote.find('Everybody')
```

[75]:
```
0
```

[76]:
```
AE_quote.find('EverybodyHello')
```

[76]:
```
-1
```

[77]:
```
AE_quote
```

[77]:
```
'Everybody is a genius.'
```

[78]:
```
AE_quote.find('genius')
```

[78]:
```
15
```

[79]:
```
len('genius')
```

[79]:
```
6
```

[80]:
```
AE_quote[15: 15 + len('genius')]
```

[80]:
```
'genius'
```

[81]:
```
AE_quote[AE_quote.find('genius'):
         AE_quote.find('genius') + len('genius')]
```

[81]:
```
'genius'
```

[82]:
```
AE_quote[AE_quote.find('Everybody') :
         (AE_quote.find('Everybody')+len("Everybody"))]
```

[82]:
```
'Everybody'
```

[83]:
```
sub_string = 'Everybody'

AE_quote[AE_quote.find(sub_string) :
         (AE_quote.find(sub_string)+len(sub_string))]
```

[83]:
```
'Everybody'
```

[84]:
```
sub_string = 'genius'

AE_quote[AE_quote.find(sub_string) :
         (AE_quote.find(sub_string)+len(sub_string))]
```

[84]:
```
'genius'
```

3.2.3 index() method

- tells us if a string 'substr' occurs in the string and return the index where the substring starts, otherwise it will raise an error.
- *Syntax:*

`str.index(substring, start, end)`

- `start` and `end`
  - integers that indicate the indices where to start and end the count
  - optional, if omitted, the whole string is inspected

[85]:
```
AE_quote.index('Everybody')
```

[85]:
```
0
```

[86]:
```
AE_quote.index('genius')
```

[86]:
```
15
```

[87]:
```
AE_quote
```

[87]:
```
'Everybody is a genius.'
```

[88]:
```
AE_quote.index('fish')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[88], line 1
----> 1 AE_quote.index('fish')

ValueError: substring not found
```

[89]:
```
AE_quote.find('fish')
```

[89]:
```
-1
```

3.2.4 strip() method

returns a copy of the string in which all characters given as argument are stripped from the beginning and the end of the string.

- Syntax:

`str.strip([chars])`

- Default argument is the space character (remove the white spaces)

[90]:
```
ER_quote = "   Great minds discuss ideas; average minds discuss events; small␣
↪minds discuss people.  "
```

[91]:
```
ER_quote
```

[91]:
```
'   Great minds discuss ideas; average minds discuss events; small minds discuss
people.  '
```

[92]:
```
ER_quote.strip()
```

[92]:
```
'Great minds discuss ideas; average minds discuss events; small minds discuss
people.'
```

[93]:
```
ER_quote_new = ER_quote.strip()
ER_quote_new
```

[93]:
```
'Great minds discuss ideas; average minds discuss events; small minds discuss
people.'
```

[94]:
```
ER_quote_new.strip('.')
```

[94]:
```
'Great minds discuss ideas; average minds discuss events; small minds discuss
people'
```

[95]:
```
ER_quote
```

[95]:
```
'   Great minds discuss ideas; average minds discuss events; small minds discuss
people.  '
```

[96]:
```
ER_quote.strip('.')
```

[96]:
```
'   Great minds discuss ideas; average minds discuss events; small minds discuss
people.  '
```

[97]:
```
ER_quote.strip('. ')
```

[97]:
```
'Great minds discuss ideas; average minds discuss events; small minds discuss
people'
```

4 Further readings

- "Built-in String Methods" in this tutorial.

4.1 Assignment: HW3

- released today
- due by 09/25

5 Next Class

- additional string methods
- list

Readings:

- Chapter 10