04.2_Strings_Iteration

September 13, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Strings
- Iteration with `for` loops

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}

2.1 Indexing String

To access each separate character in a string

Structure: `string[index]` * string variable name * square brackets * index: integer (starts from 0 in python)

2.1.1 Access the last charater in a string

- Find the index of the last charater
  - A built-in function called `len()` that gives the information about length of an object
- Use that index to access the character

[5]:
```
my_string = "Hello World"
```

[6]:
```
my_string[len(my_string)-1]
```

[6]:
```
'd'
```

2.1.2 Negative index

Another way to grab the last element so we don't need to calculate the length and substract one.

Count backwards!

[7]:
```
my_string[-1]
```

[7]:
```
'd'
```

[8]:
```
my_string[-2]
```

[8]:
```
'l'
```

2.1.3 Index has to be an integer!

Data type matters

[9]:
```
my_string[1.0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[9], line 1
----> 1 my_string[1.0]

TypeError: string indices must be integers, not 'float'
```

[10]:
```
my_string[int(1.0)]
```

[10]:
```
'e'
```

[11]:
```
my_string
```

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

[12]:
```
my_string[len(my_string) - 1.0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[12], line 1
----> 1 my_string[len(my_string) - 1.0]

TypeError: string indices must be integers, not 'float'
```

[13]:
```
type(len(my_string)- 1.0 )
```

[13]:
```
float
```

2.1.4 strings are immutable

a string value cannot be updated

[14]:
```
my_string
```

[14]:
```
'Hello World'
```

[15]:
```
my_string[5]
```

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

[16]:
```
my_string[5] = "_"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 my_string[5] = "_"

TypeError: 'str' object does not support item assignment
```

[17]:
```
'Hello'+'_'+'World'
```

[17]:
```
'Hello_World'
```

[18]:
```
'Hello_World'
```

[18]:
```
'Hello_World'
```

2.1.5 `in` operator

- Check whether a substring occurs in the string
- Returns a boolean value

[19]:
```
my_string
```

[19]:
```
'Hello World'
```

[20]:
```
"d" in my_string
```

[20]:
```
True
```

[21]:
```
"hello" in my_string
```

[21]:
```
False
```

[22]:
```
"Hello" in my_string
```

[22]:
```
True
```

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

The idea of *iteration* (in plain English) is to repeat a process several times. If you have any programming experience with another language (like C or Java, say), you may have an idea of how to create iteration with `for` statements. But these are a little different in Python, as you can read in the documentation.

A Python `for` statement iterates over the items of a sequence, naturally.

[23]:
```
my_string
```

[23]:
```
'Hello World'
```

[26]:
```
for s in my_string:
    print(s)
```

```
H
e
l
l
o
```

```
W
o
r
l
d
```

[33]:
```
for s in my_string:
    if s == " ":
        break
    print(s)
```

```
H
e
l
l
o
```

[34]:
```
print(s)
```

[35]:
```
iteration = 0
for s in my_string:
    print(s)
    iteration = iteration + 1
    if iteration == 5:
        break
```

```
H
e
l
l
o
```

[36]:
```
print(s)
```

```
o
```

[37]:
```
i
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[37], line 1
----> 1 i

NameError: name 'i' is not defined
```

[38]:
```
for i in my_string:
    print(i)
```

```
H
e
l
l
o

W
o
r
l
d
```

[39]:
```
i
```

[39]:
```
'd'
```

3.0.1 Syntax of a `for` statement

```
for s in my_string:
    print(s)
```

- `for`: keyword for for Loops (repetitions)
- `in`: operator
  - check whether a specified value is a constituent element of a sequence like string, array, list, or tuple etc.
  - `s in my_string`: check whether `s` is a constituent element of `my_string`
- logic:
  - assign the first element of `my_string` to `s`, execute the block that follows.
  - assign the second element of `my_string` to `s`, execute the block that follows.
  - ...
  - assign the last element of `my_string` to `s`, execute the block that follows.

3.0.2 Group Exercise

Write a `for` statement to find each element in the string `"python is fun!"` and add a suffix `"_suffix"` to each element and print it out. For instance, the first printed out string is `"p_suffix"`

When you are done, raise your hand!

[40]:
```
s = "python is fun!"
for i in s:
    print(i + "_suffix")
```

```
p_suffix
y_suffix
t_suffix
```

```
h_suffix
o_suffix
n_suffix
 _suffix
i_suffix
s_suffix
 _suffix
f_suffix
u_suffix
n_suffix
!_suffix
```

[41]:
```
s = "python is fun!"
for i in s:
    if i != " ":
        print(i + "_suffix")
```

```
p_suffix
y_suffix
t_suffix
h_suffix
o_suffix
n_suffix
i_suffix
s_suffix
f_suffix
u_suffix
n_suffix
!_suffix
```

[42]:
```
s = "python is fun!"
for i in s:
    if i == " ":
        continue
    print(i + "_suffix")
```

```
p_suffix
y_suffix
t_suffix
h_suffix
o_suffix
n_suffix
i_suffix
s_suffix
f_suffix
u_suffix
n_suffix
!_suffix
```

3.0.3 `for/range` statements

Can be used to iterate over sequences (e.g, a string) using an index

- range(): a built-in function that provides a sequence of integers

```
for i in range(3):
    print(i)
```

[44]:
```
range(3)
```

[44]:
```
range(0, 3)
```

[45]:
```
list(range(3))
```

[45]:
```
[0, 1, 2]
```

[46]:
```
for i in range(3):
    print(i)
```

```
0
1
2
```

[47]:
```
a = "UNT"
```

[48]:
```
a
```

[48]:
```
'UNT'
```

[49]:
```
a[0]
```

[49]:
```
'U'
```

[50]:
```
a[1]
```

[50]:
```
'N'
```

[51]:
```
a[2]
```

[51]:
```
'T'
```

[52]:
```
a
```

[52]:
```
'UNT'
```

[53]:
```
for i in range(3):
    print(a[i])
```

```
U
N
T
```

[54]:
```
a = "UNT"
b = "UNC"
for i in range(3):
    print(a[i] +"-"+b[i])
```

```
U-U
N-N
T-C
```

[55]:
```
a = "UNT"
b = "UNC"
for i in range(3):
    print(a[i] +"-"+b[-i-1])
```

```
U-C
N-N
T-U
```

[56]:
```
for i in range(len(a)):
    print(a[i])
```

```
U
N
T
```

3.0.4 for/range statements

Can be used to iterate over sequences (e.g, a string) using an index

- find the length of the string
- generate a sequence of integers (representing indexes)
- get the character using indexing

[57]:
```
a = "python is fun!"
```

[58]:
```
length_a = len(a)
length_a
```

[58]:
```
14
```

[59]:
```
range(len(a))
```

[59]:
```
range(0, 14)
```

[60]:
```
list(range(len(a)))
```

[60]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

[61]:
```
for i in range(14):
    print(a[i])
```

```
p
y
t
h
o
n

i
s

f
u
n
!
```

[62]:
```
for c in a:
    print(c)
```

```
p
y
t
h
o
n

i
s

f
u
n
!
```

3.0.5 Group Exercise

Write a `for/range` statement to print each element in the string `"It is a great day!"`:

When you are done, raise your hand!

[63]:
```
str_now = "It is a great day!"
for i in str_now:
    print(i)
```

```
I
```

```
t

i
s

a

g
r
e
a
t

d
a
y
!
```

[64]:
```
str_now = "It is a great day!"
for i in range(len(str_now)):
    print(str_now[i])
```

```
I
t

i
s

a

g
r
e
a
t

d
a
y
!
```

4 Next Class

- String methods

Readings: Chapter 9