



Python Quick Tips

Working with Strings



Python Quick Tips #4

Working with Strings

Strings



In part 1, we saw that **strings** are a data type

What are Strings

Output

String, 1, 5.4

Syntax

"String", "1", "5.4"

Or

'String', '1', '5.4'

Convert to String

Function `str(object)`

test.py ×

```
1 str1 = str(1)
2 str2 = str(2.5)
3 str3 = str(True)
4
5 print([str1, str2, str3])
```

Shell ×

```
>>> %Run test.py
['1', '2.5', 'True']
```

' vs. "

Be mindful of characters ' and " in strings

```
test.py ×
1 print("stri'ng")

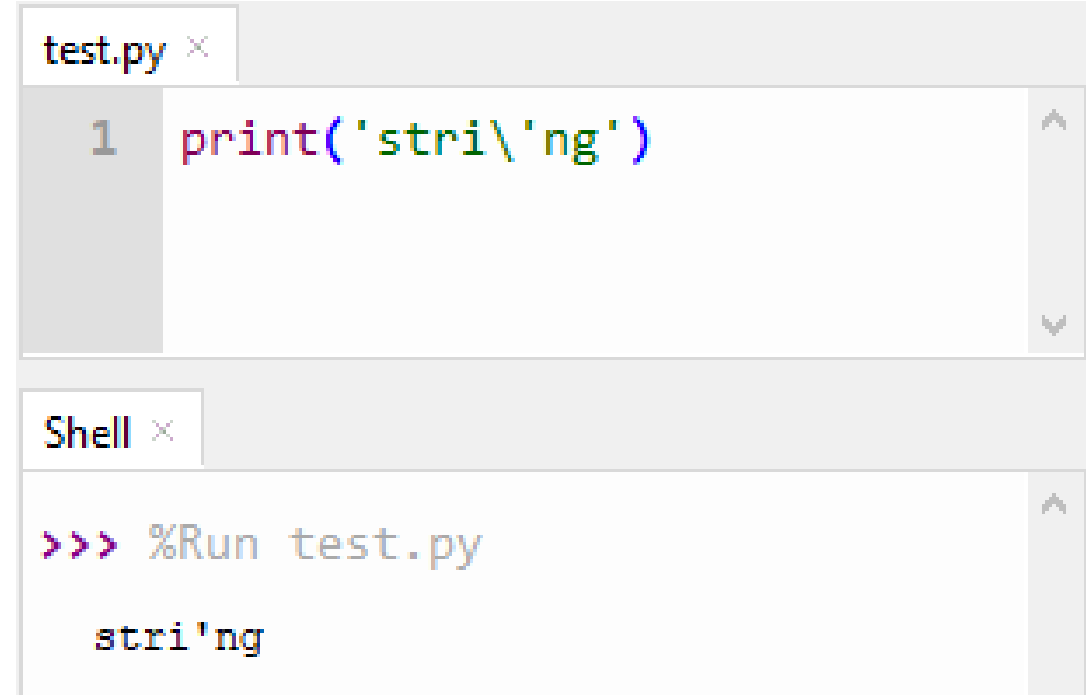
Shell ×
>>> %Run test.py
stri'ng
>>> |
```

```
test.py ×
1 print('stri'ng')

Shell ×
>>> %Run test.py
Traceback (most recent call last):
  File "C:\Users\Gavin\Desktop\test.py", line 1
    print('stri'ng')
            ^
SyntaxError: invalid syntax
>>> |
```

Escape Sequences

Syntax
'string\'string'



The image shows a screenshot of a code editor and a shell window. The code editor has a tab labeled 'test.py' and contains the following code:

```
1 print('stri\'ng')
```

Below the code editor is a shell window with a tab labeled 'Shell'. It shows the command to run the Python file and the output:

```
>>> %Run test.py  
stri'ng
```

Escape Sequences

There are more types of
Escape sequences

Escape-sequence	Purpose
<code>\n</code>	New line
<code>\\</code>	Backslash character
<code>\'</code>	Apostrophe '
<code>\"</code>	Quotation mark "
<code>\a</code>	Sound signal
<code>\b</code>	Slaughter (backspace key symbol)
<code>\f</code>	The conversion of format
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\v</code>	Vertical tab
<code>\xhh</code>	Character with hex code hh
<code>\ooo</code>	Character with octal value ooo
<code>\0</code>	Character Null (not a string terminator)
<code>\N{id}</code>	Identifier ID of Unicode database
<code>\uhhhh</code>	16-bit Unicode character in hexadecimal format
<code>\Uhhhhhhhh</code>	32-bit Unicode character in hexadecimal format
<code>\другое</code>	Not an escape sequence (\ character is stored)

Raw Strings

Syntax r"string"

```
test.py ×
1 path = 'C:\Users\Gavin\Desktop'
2
3 print(path)

Shell ×
>>> %Run test.py
Traceback (most recent call last):
  File "C:\Users\Gavin\Desktop\test.py", line 1
    path = 'C:\Users\Gavin\Desktop'
           ^
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 2-3: truncated \UXXXXXXXX escape

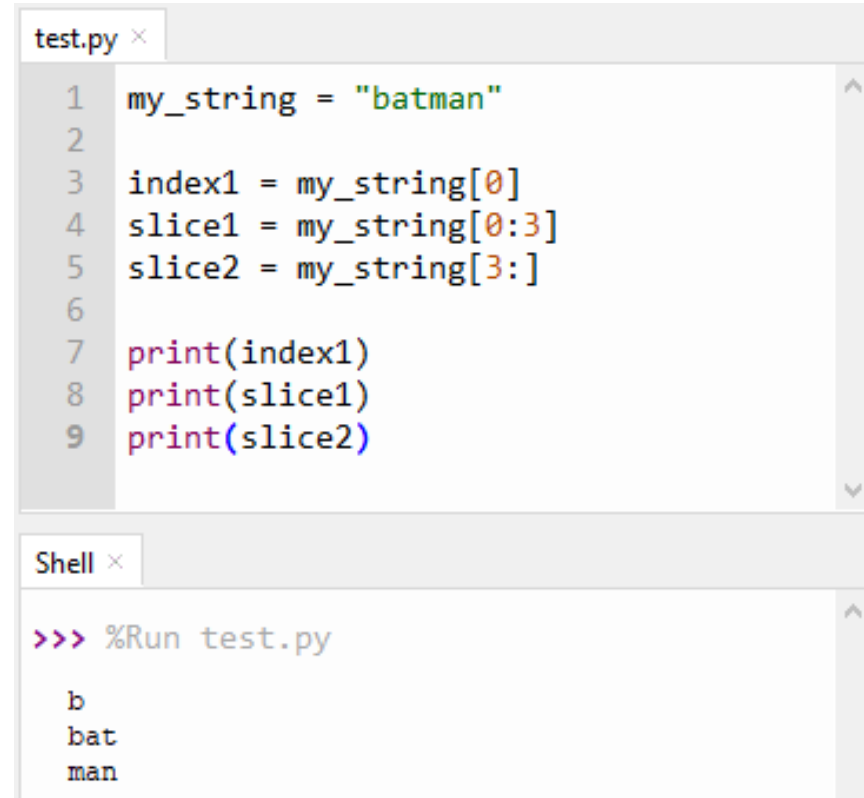
>>> |
```

```
test.py ×
1 path = r"C:\Users\Gavin\Desktop"
2 print(path)

Shell ×
>>> %Run test.py
C:\Users\Gavin\Desktop
>>> |
```

Slicing and Indexing

Syntax
string[index], string[s:e]



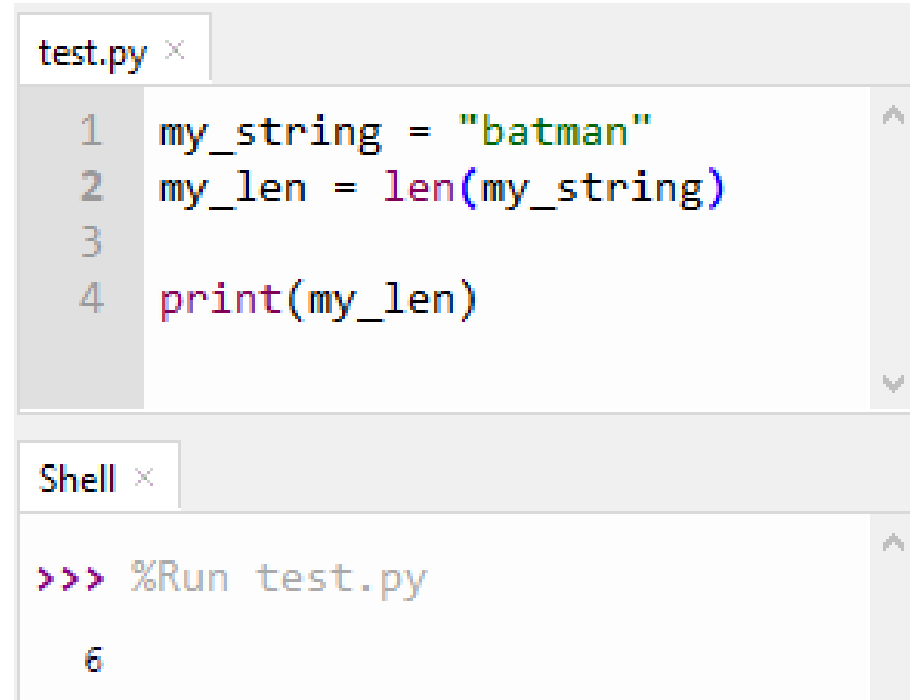
```
test.py x
1 my_string = "batman"
2
3 index1 = my_string[0]
4 slice1 = my_string[0:3]
5 slice2 = my_string[3:]
6
7 print(index1)
8 print(slice1)
9 print(slice2)

Shell x
>>> %Run test.py

b
bat
man
```

Length

Function
len(string)

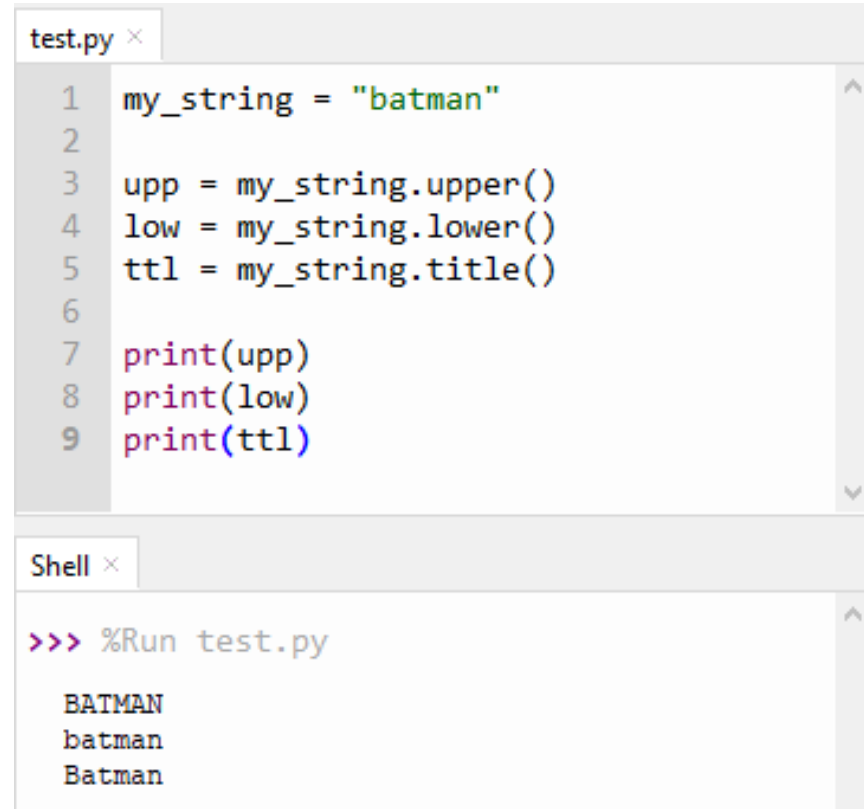


The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains a script with four lines of code: line 1 assigns 'batman' to 'my_string', line 2 uses 'len()' to calculate the length and store it in 'my_len', line 3 is empty, and line 4 prints 'my_len'. The bottom panel, titled 'Shell', shows the command '%Run test.py' being executed, followed by the output '6'.

```
test.py ×  
1 my_string = "batman"  
2 my_len = len(my_string)  
3  
4 print(my_len)  
  
Shell ×  
  
>>> %Run test.py  
  
6
```

Case

Methods
string.upper() etc.



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

```
1 my_string = "batman"
2
3 upp = my_string.upper()
4 low = my_string.lower()
5 ttl = my_string.title()
6
7 print(upp)
8 print(low)
9 print(ttl)
```

The bottom panel, titled 'Shell', shows the command prompt and the output of the script:

```
>>> %Run test.py
BATMAN
batman
Batman
```

Reverse

Syntax
`string[::-1]`



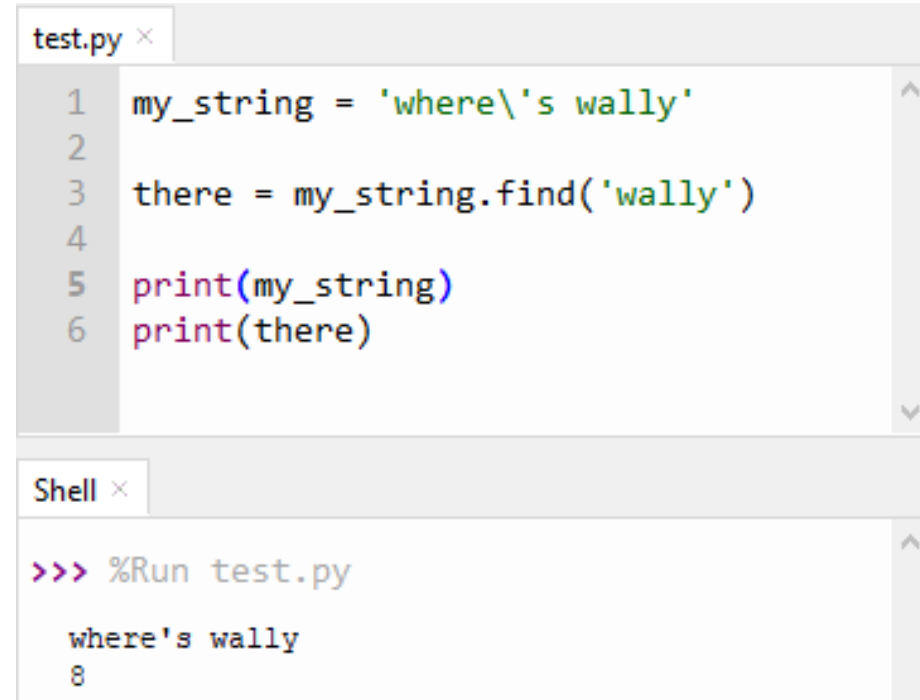
The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

```
1 my_hero = 'batman'
2
3 my_oreh = my_hero[::-1]
4
5 print(my_oreh)
```

The bottom panel, titled 'Shell', shows the command prompt with the command `>>> %Run test.py` and the output `namtab`.

Find

Method
`string.find(substring, s, e)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

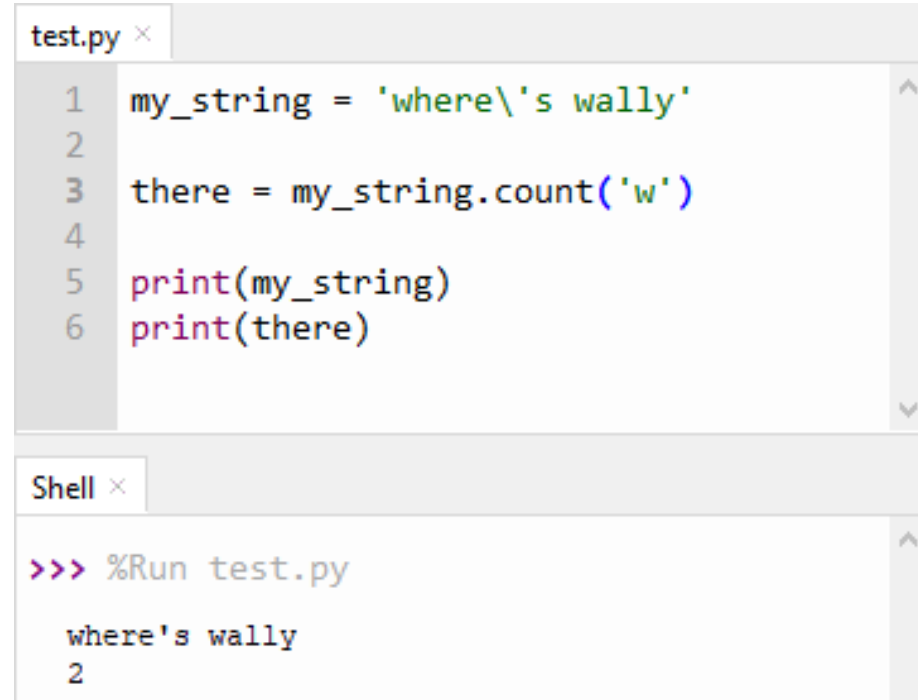
```
1 my_string = 'where\'s wally'
2
3 there = my_string.find('wally')
4
5 print(my_string)
6 print(there)
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
where's wally
8
```

Count

Method
`string.find(substring, s, e)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

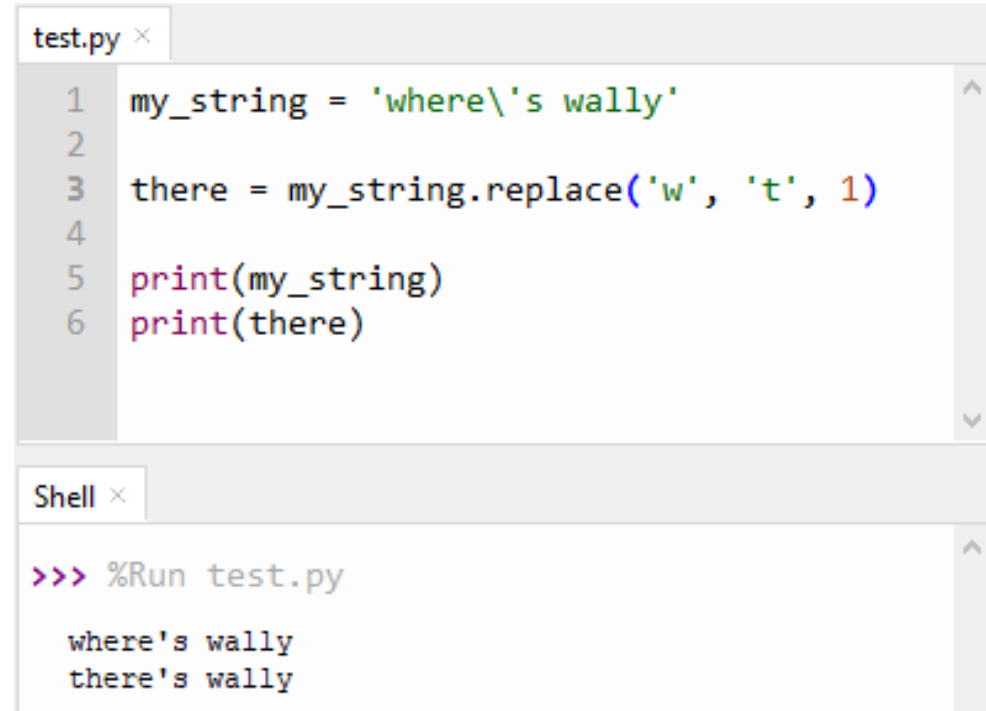
```
1 my_string = 'where\'s wally'
2
3 there = my_string.count('w')
4
5 print(my_string)
6 print(there)
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
where's wally
2
```

Replace

Method
`string.replace(find, rep, c)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains a script with six lines of code. The bottom panel, titled 'Shell', shows the output of running the script. The script defines a string 'where's wally', creates a new string 'there's wally' by replacing the first 'w' with 't', and prints both strings. The shell output shows the original string followed by the modified string.

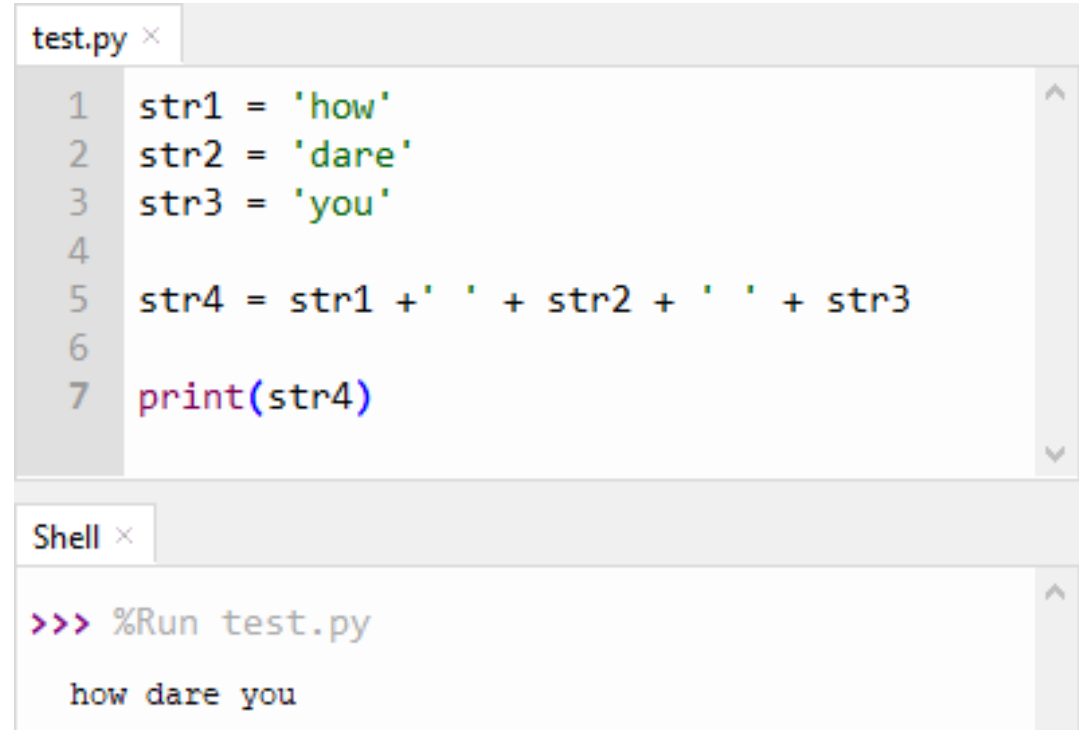
```
test.py ×
1 my_string = 'where's wally'
2
3 there = my_string.replace('w', 't', 1)
4
5 print(my_string)
6 print(there)
```

```
Shell ×
>>> %Run test.py
where's wally
there's wally
```


Add

Syntax

string + string



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

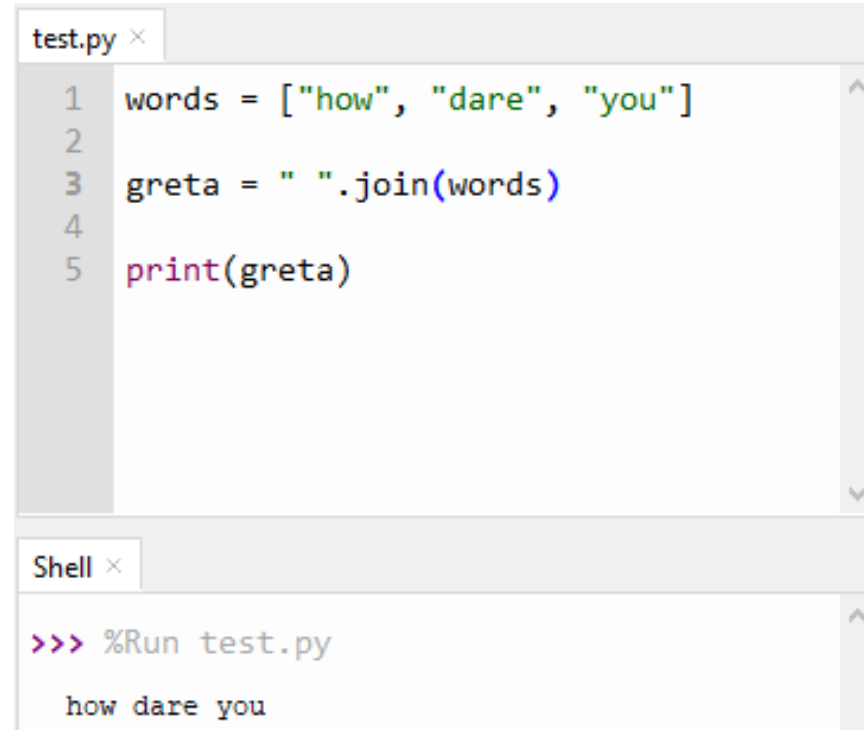
```
1 str1 = 'how'
2 str2 = 'dare'
3 str3 = 'you'
4
5 str4 = str1 + ' ' + str2 + ' ' + str3
6
7 print(str4)
```

The bottom panel, titled 'Shell', shows the command prompt output:

```
>>> %Run test.py
how dare you
```

Join

Method
`separator.join(list)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

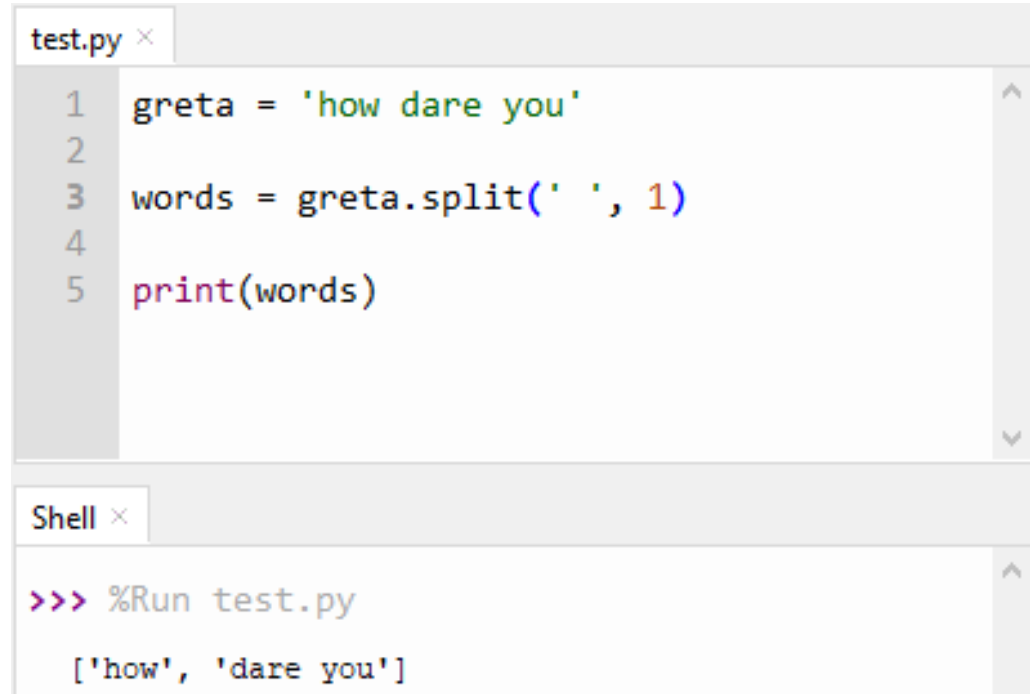
```
1 words = ["how", "dare", "you"]
2
3 greta = " ".join(words)
4
5 print(greta)
```

The bottom panel, titled 'Shell', shows the execution of the script:

```
>>> %Run test.py
how dare you
```

Split

Method
`string.split(sep, c)`



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains a script that splits the string 'how dare you' into a list. The bottom panel, titled 'Shell', shows the execution of the script, resulting in the output ['how', 'dare you'].

```
test.py ×  
1 greta = 'how dare you'  
2  
3 words = greta.split(' ', 1)  
4  
5 print(words)  
  
Shell ×  
  
>>> %Run test.py  
['how', 'dare you']
```

rSplit

Method
`string.rsplitlep, c)`

```
test.py ×
1 greta = 'how dare you'
2
3 words = greta.rsplitlep, 1)
4
5 print(words)

Shell ×
>>> %Run test.py
['how dare', 'you']
```

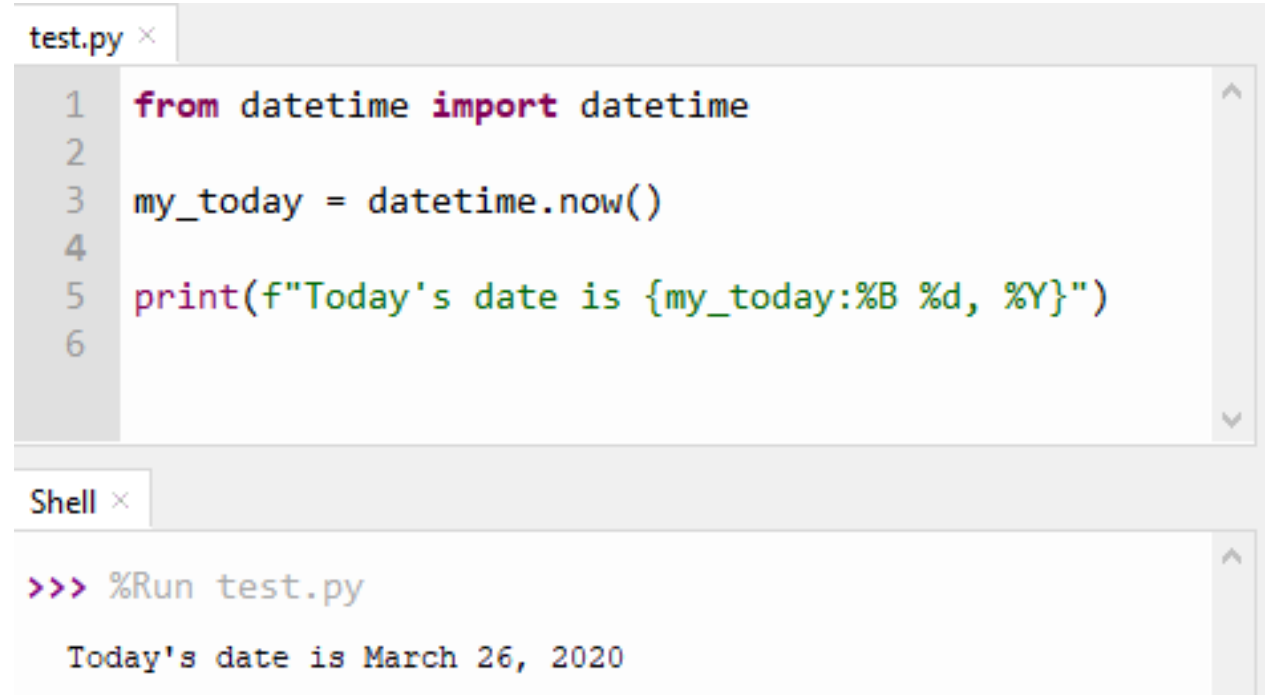
Strip

Method
`string.(l/r)strip(sub)`

```
test.py ×  
1 messy = '    string  '  
2  
3 clean = messy.strip()  
4 rclean = messy.rstrip()  
5 lclean = messy.lstrip()  
6  
7 print(messy + '.')  
8 print(clean + '.')  
9 print(rclean + '.')  
10 print(lclean + '.')  
  
Shell ×  
  
>>> %Run test.py  
  
    string .  
string.  
    string.  
string .
```

f Strings

Syntax
`f"string{var: format}"`



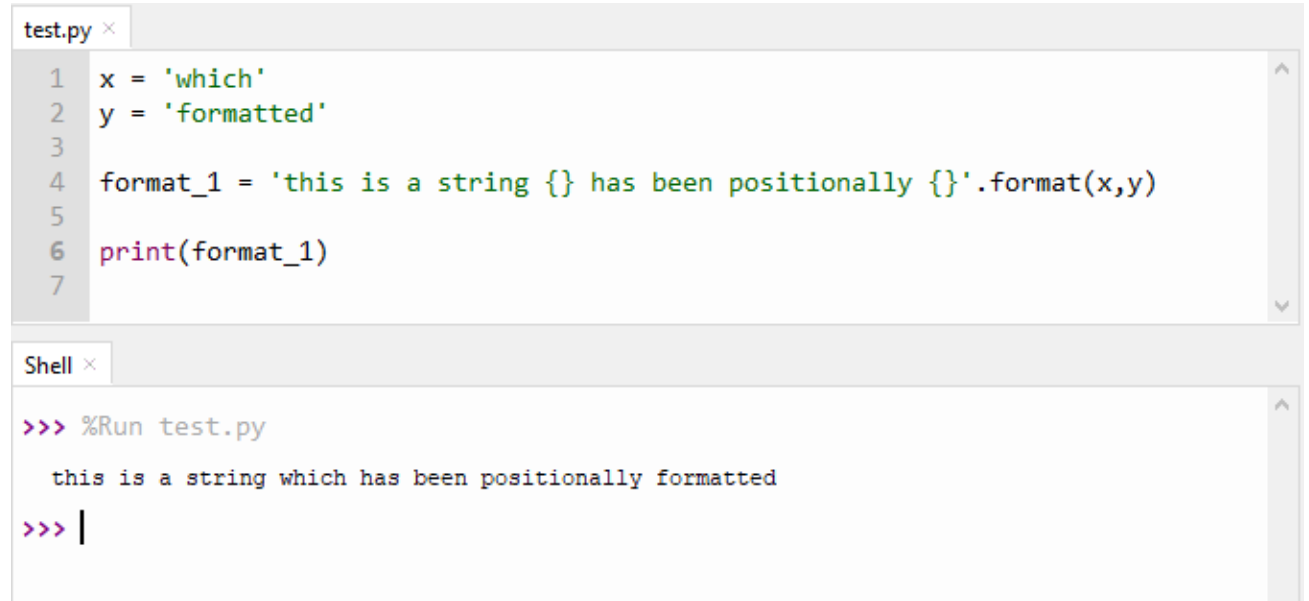
The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains a script that imports the datetime module, gets the current date and time, and prints it using an f-string. The bottom panel, titled 'Shell', shows the command to run the script and the resulting output.

```
test.py ×  
1 from datetime import datetime  
2  
3 my_today = datetime.now()  
4  
5 print(f"Today's date is {my_today:%B %d, %Y}")  
6  
  
Shell ×  
  
>>> %Run test.py  
Today's date is March 26, 2020
```

Positional Formatting

Syntax

“string{var}”.format(var)



The image shows a code editor window titled 'test.py' with the following Python code:

```
1 x = 'which'
2 y = 'formatted'
3
4 format_1 = 'this is a string {} has been positionally {}'.format(x,y)
5
6 print(format_1)
7
```

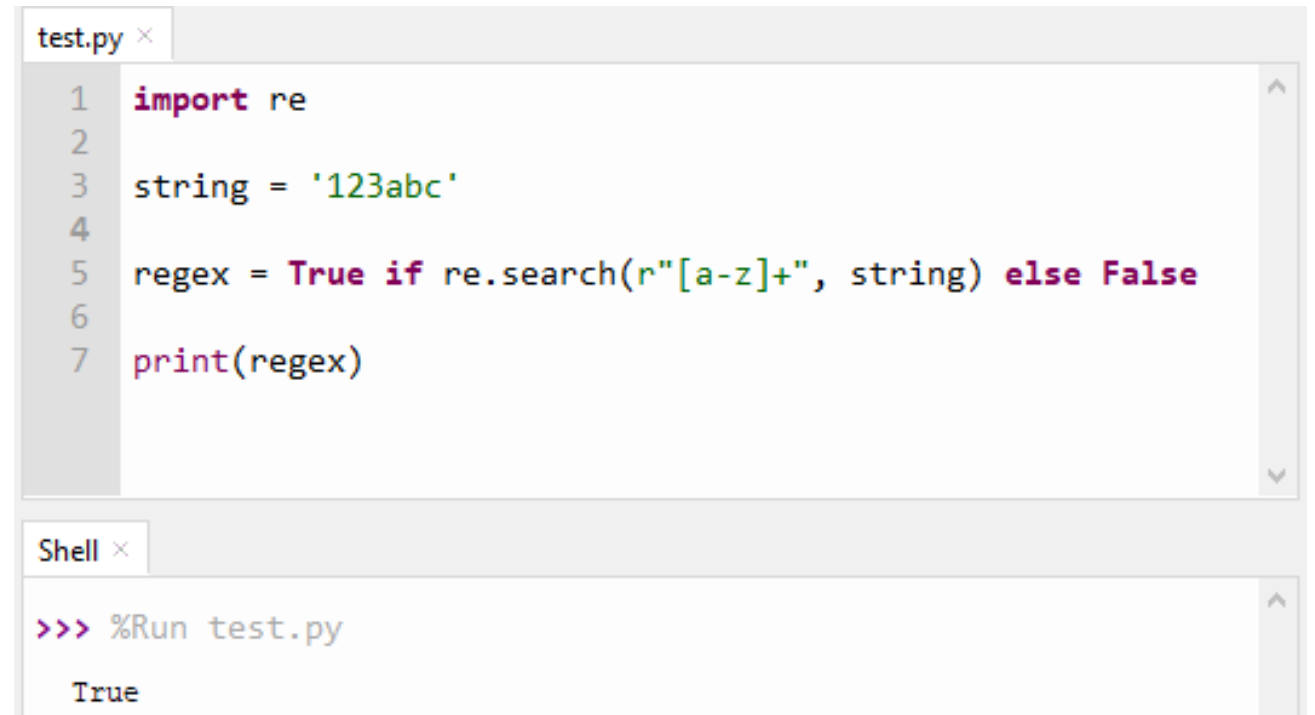
Below the code editor is a shell window titled 'Shell' showing the execution of the script:

```
>>> %Run test.py
    this is a string which has been positionally formatted
>>> |
```

Regular Expressions

Syntax

```
re.match(r"regex", string)  
re.search(r"regex", string)
```



The screenshot shows a Python IDE with two panels. The top panel, titled 'test.py', contains the following code:

```
1 import re  
2  
3 string = '123abc'  
4  
5 regex = True if re.search(r"[a-z]+", string) else False  
6  
7 print(regex)
```

The bottom panel, titled 'Shell', shows the command prompt and the output of running the script:

```
>>> %Run test.py  
True
```




Next on #5
Defining Functions



Python Quick Tips

Working with Strings