Strings

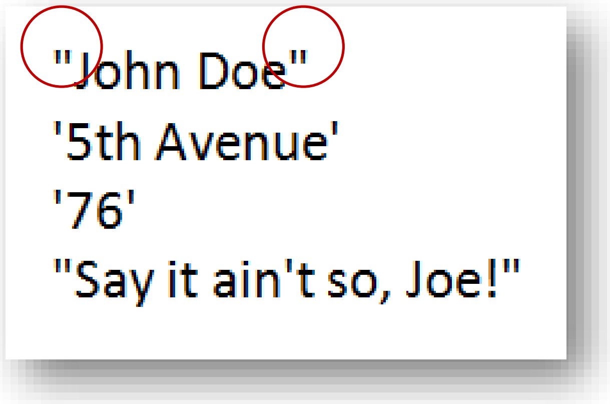
Section 2

Chapter 2

Quiz 3

String Literals

- A **string literal** is a sequence of characters that is treated as a single item.
- Written as a sequence of characters surrounded by *either single quotes (') or double quotes (").*



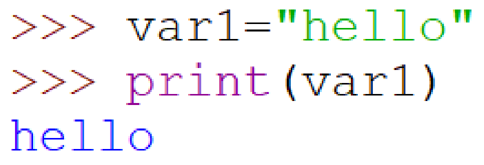
```
"John Doe"
'5th Avenue'
'76'
"Say it ain't so, Joe!"
```

Opening and closing quotation marks must be the same type

String Variables

- Variables also can be assigned string values

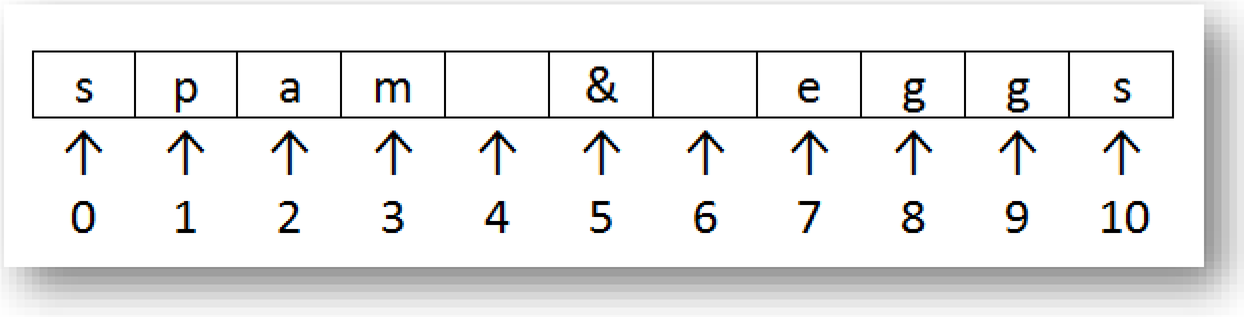
- Print a string
  - Quotation marks not included in display



```
>>> var1="hello"
>>> print(var1)
hello
```

Indices and Slices

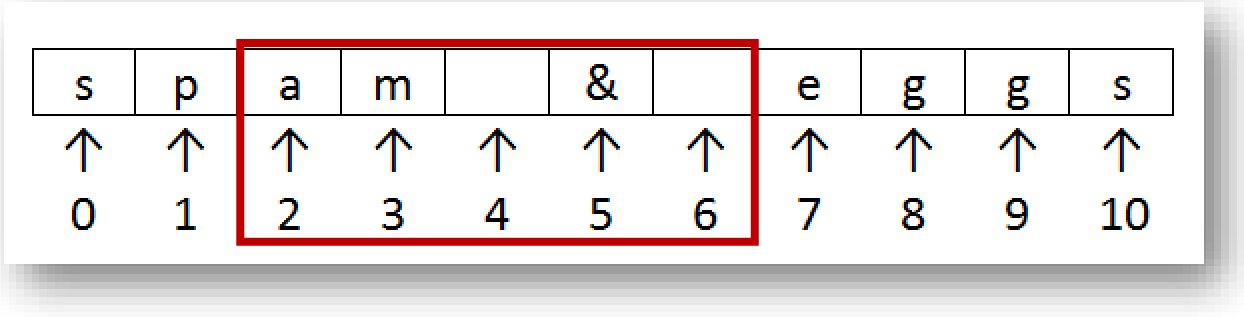
- Position or index of a character in a string
  - Identified with one of the numbers **0**, **1**, **2**, **3**, ...



```
>>> school="CityU"
>>> school[0]
'C'
>>> school[1]
'i'
>>> school[2]
't'
>>> school[3]
'y'
>>> school[4]
'U'
```

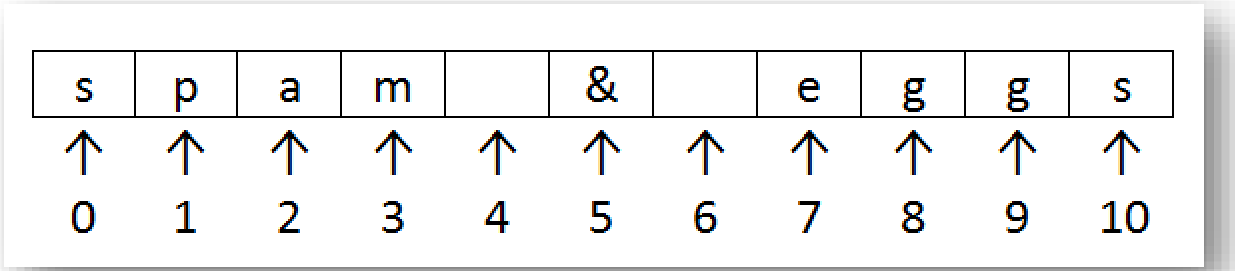
Indices and Slices

- If str1 is a string, then **`str1[m:n]`** is the substring or slice beginning at position m and ending at position n - 1
  - Example “spam & eggs”[2:7]



```
>>> str1="spam & eggs"
>>> str1[2:7]
'am & '
```

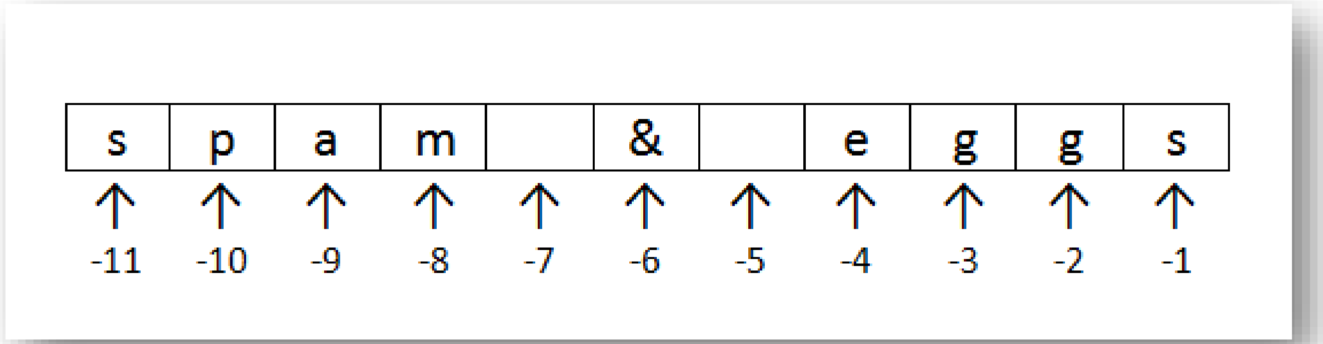
Indices and Slices



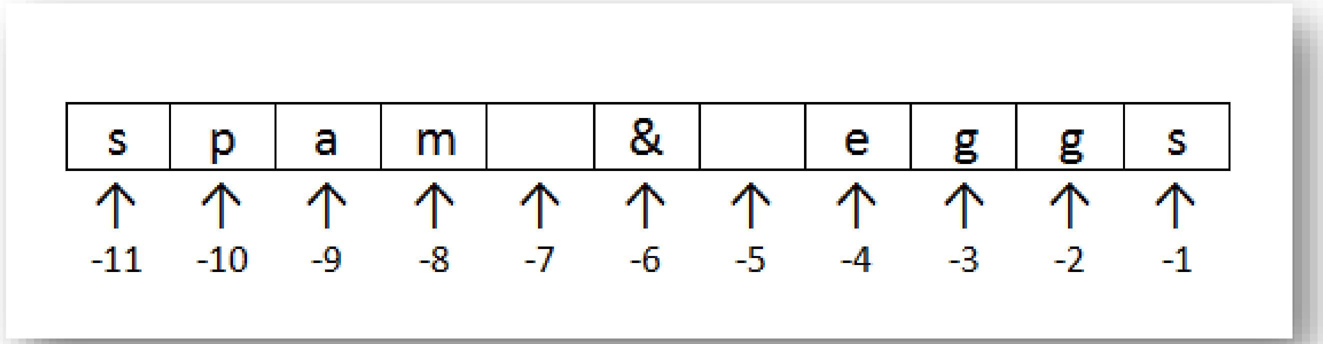
```
>>> str1="spam & eggs"
>>> str1[0]
's'
>>> str1.find('p')
1
>>> str1.find('g')
8
>>> str1.rfind('g')
9
>>> str1.find('gs')
9
>>> str1.rfind('gs')
9
>>> str1.find('huh')
-1
```

Negative Indices

- Python allows strings to be indexed by their position with regards to the right
  - Use negative numbers for indices, starting from -1.



Negative Indices

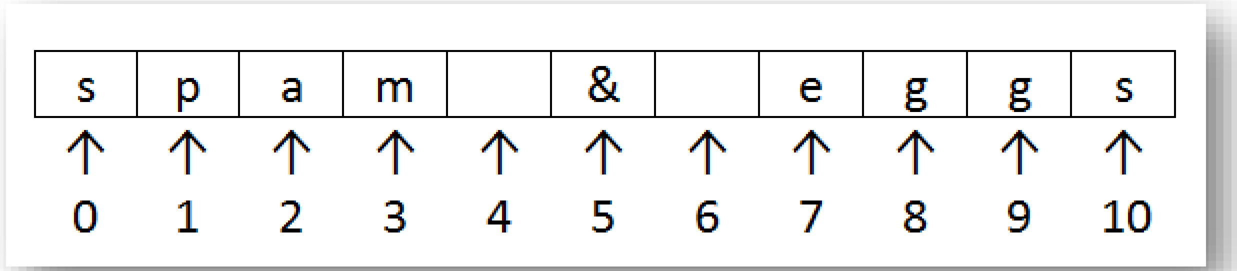


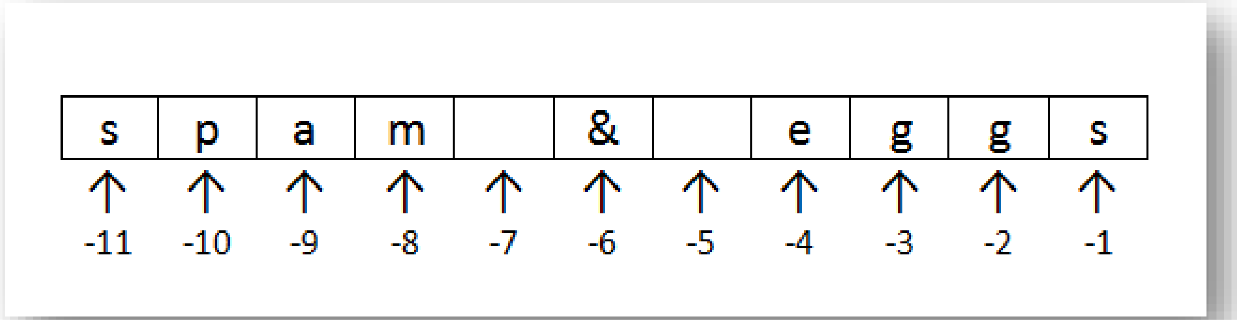
```
>>> str1="spam & eggs"
>>> str1[-1]
's'
>>> str1[-5:-2]
' eg'
>>> str1[0:-1]
'spam & egg'
```

Notice that the last character "s" is not included.

If str1 is a string, then str1[m:n] is the substring beginning at position m and ending at position n – 1. In this case, $-1 - 1 = -2$.

Default Bounds for Slices



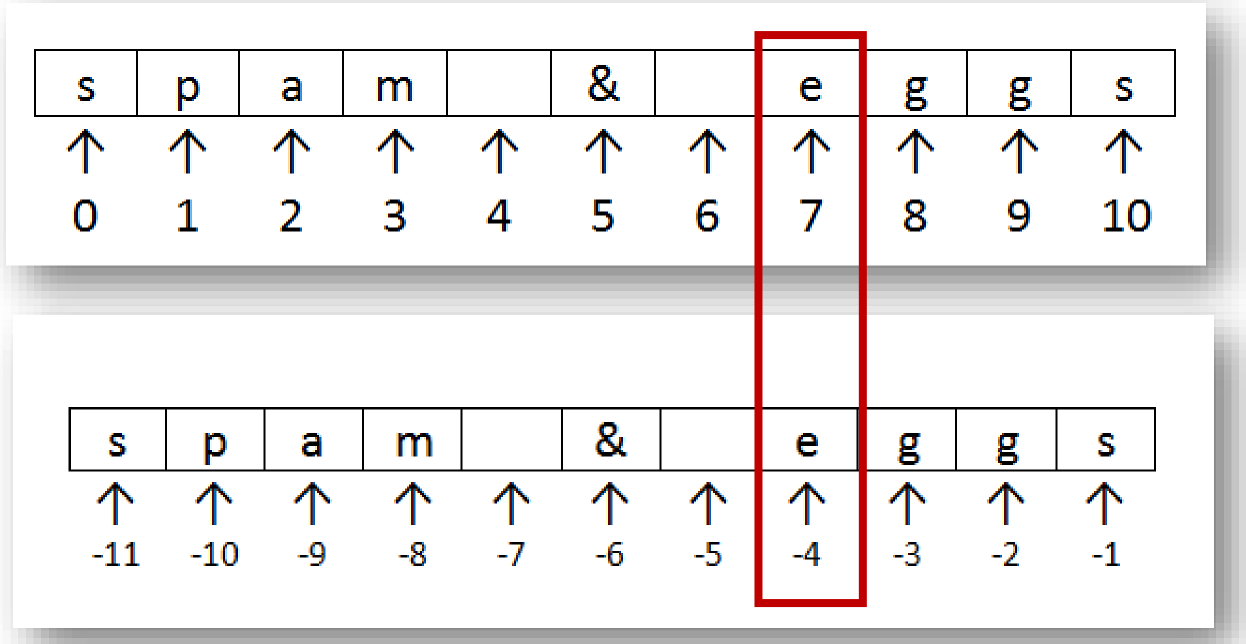


```
>>> str1="spam & eggs"
>>> str1[2:]
'am & eggs'
>>> str1[:8]
'spam & e'
>>> str1[:]
'spam & eggs'
>>> str1[-3:]
'ggs'
>>> str1[:-3]
'spam & e'
```

Positive indices start from 0, but not negative indices… Why?

- **Negative Index = Positive Index – Length**

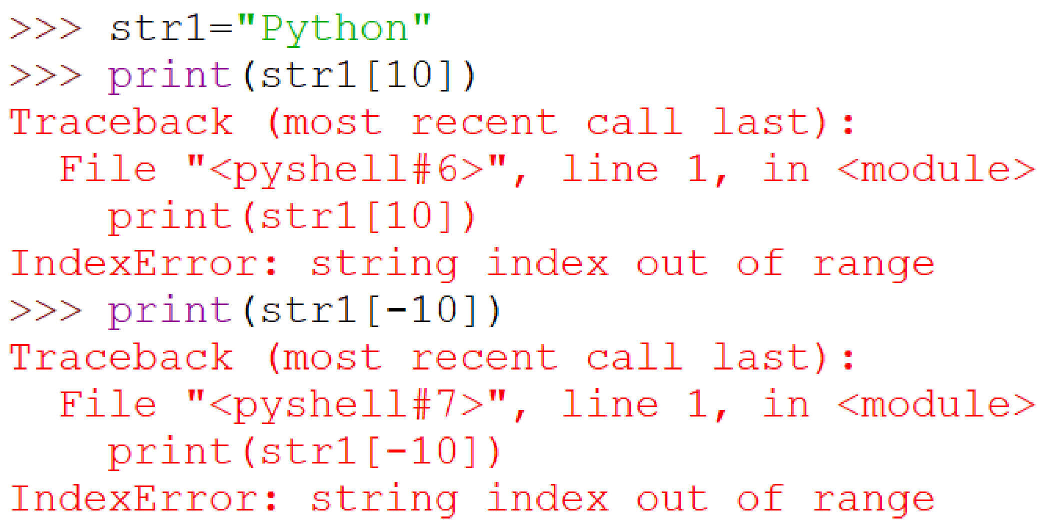
- For "e", its negative index is $7 - 11 = -4$



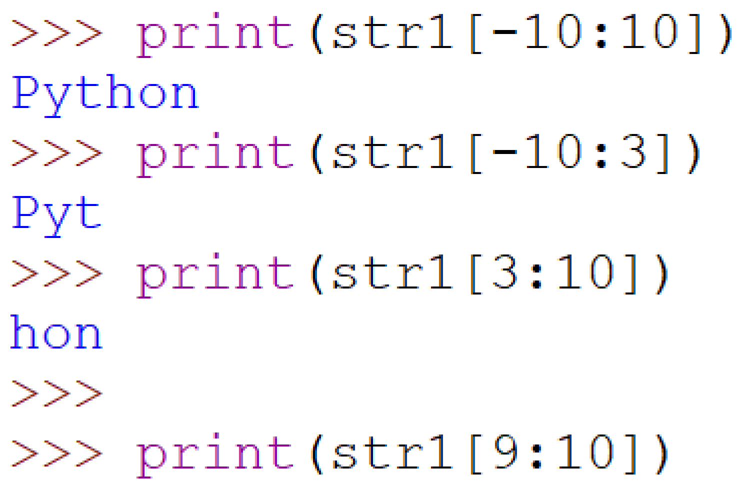
```
>>> str1="spam & eggs"
>>> str1.find("e")-len(str1)
-4
```

Indexing and Slicing Out of Bounds

- Python does not allow out of bounds indexing for individual characters of strings
  - Does allow out of bounds indices for slices



```
>>> str1="Python"
>>> print(str1[10])
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(str1[10])
IndexError: string index out of range
>>> print(str1[-10])
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print(str1[-10])
IndexError: string index out of range
```



```
>>> print(str1[-10:10])
Python
>>> print(str1[-10:3])
Pyt
>>> print(str1[3:10])
hon
>>>
>>> print(str1[9:10])
```

Slicing with skips

```
>>> ref="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
>>> ref[ : : 2] #The third entry allows the slicing to skip over some letters.
'ACEGIKMOQSUWY'
>>> ref[ 1:25: 3]
'BEHKNQTW'
>>> ref[ : : -2]  #reverse skipping
'ZXVTRPNLJHFDB'
>>> ref[ : : -1]   #this reverses the original string.
'ZYXWVUTSRQPONMLKJIHGFEDCBA'
```

String Concatenation

- Two strings can be combined to form a new string
  - Consisting of the strings joined together
  - Represented by a plus sign

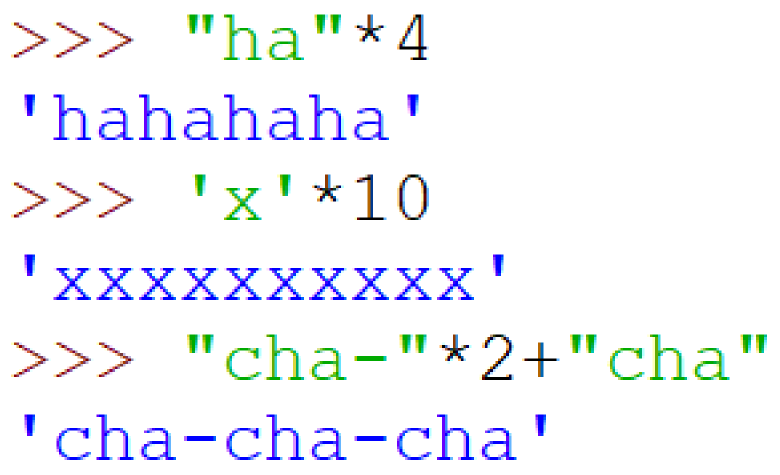
```
>>> "Hello "+"world!"
'Hello world!'
```

- Combination of strings, plus signs, functions, and methods can be evaluated
  - Called a string expression

```
>>> "a"+"b"
'ab'
>>> "b"+"a"  #string addition is not commutative
'ba'
```

String Repetition

- Asterisk operator can be used with strings to repeatedly concatenate a string with itself



```
>>> "ha"*4
'hahahaha'
>>> 'x'*10
'xxxxxxxxxx'
>>> "cha-"*2+"cha"
'cha-cha-cha'
```

String Functions and Methods

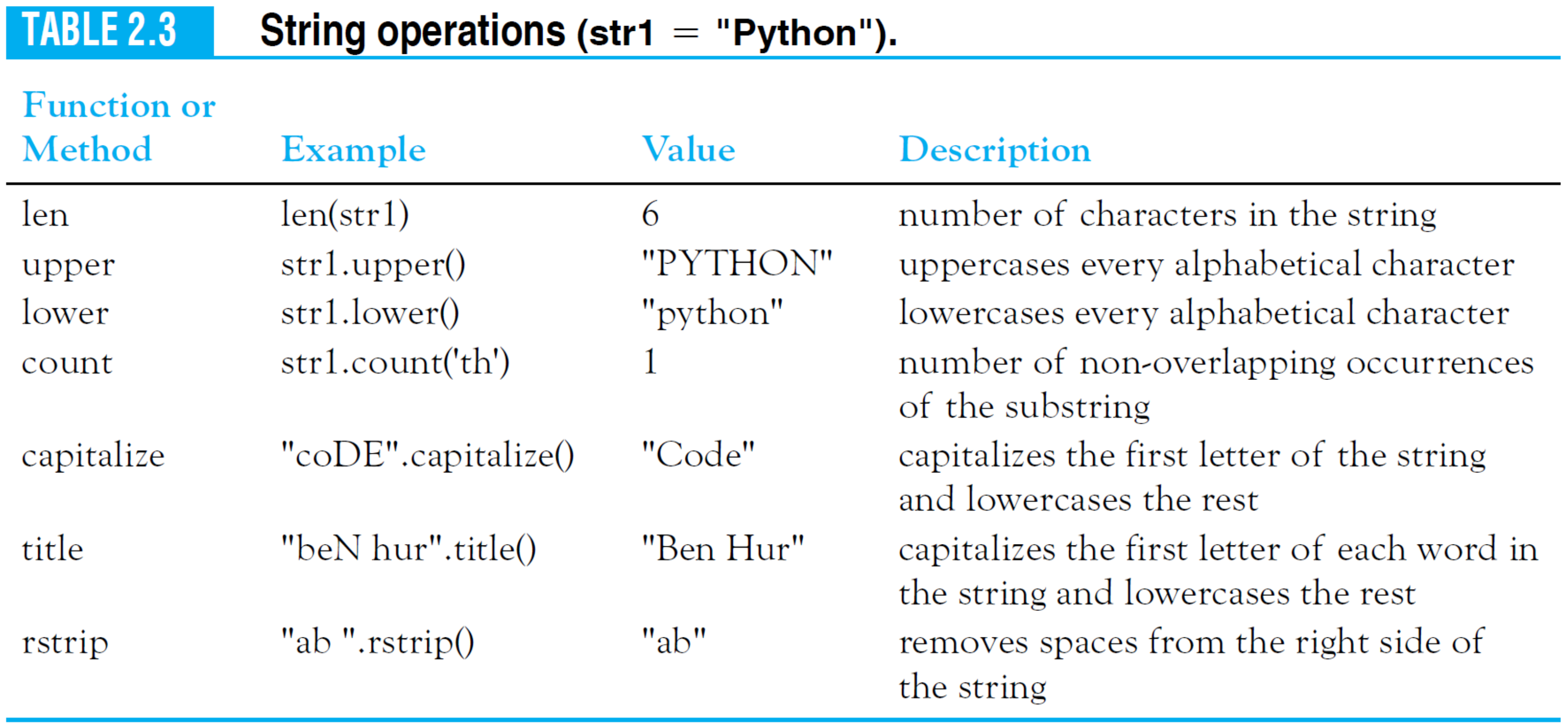
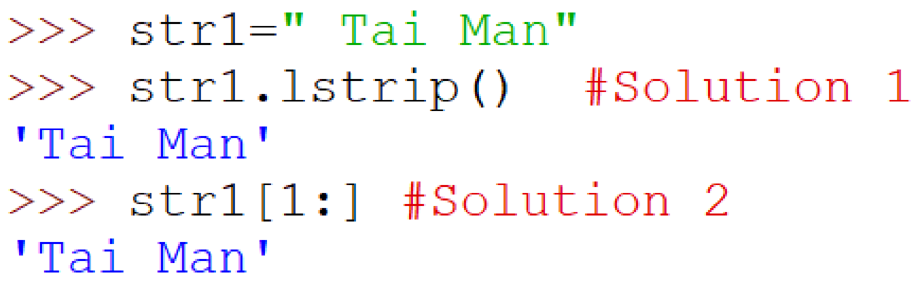


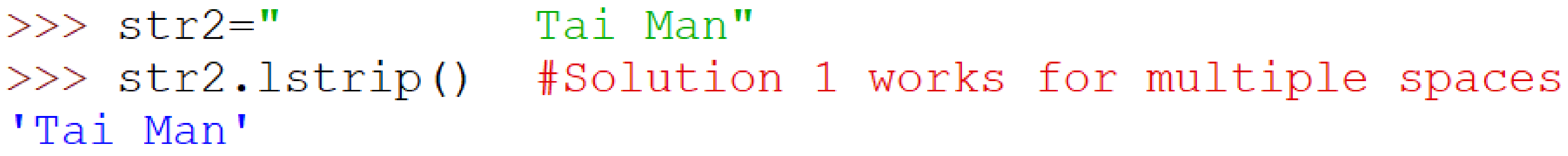
TABLE 2.3 **String operations (str1 = "Python").**

Function or Method	Example	Value	Description
len	len(str1)	6	number of characters in the string
upper	str1.upper()	"PYTHON"	uppercases every alphabetical character
lower	str1.lower()	"python"	lowercases every alphabetical character
count	str1.count('th')	1	number of non-overlapping occurrences of the substring
capitalize	"coDE".capitalize()	"Code"	capitalizes the first letter of the string and lowercases the rest
title	"beN hur".title()	"Ben Hur"	capitalizes the first letter of each word in the string and lowercases the rest
rstrip	"ab ".rstrip()	"ab"	removes spaces from the right side of the string

How to remove unnecessary spaces



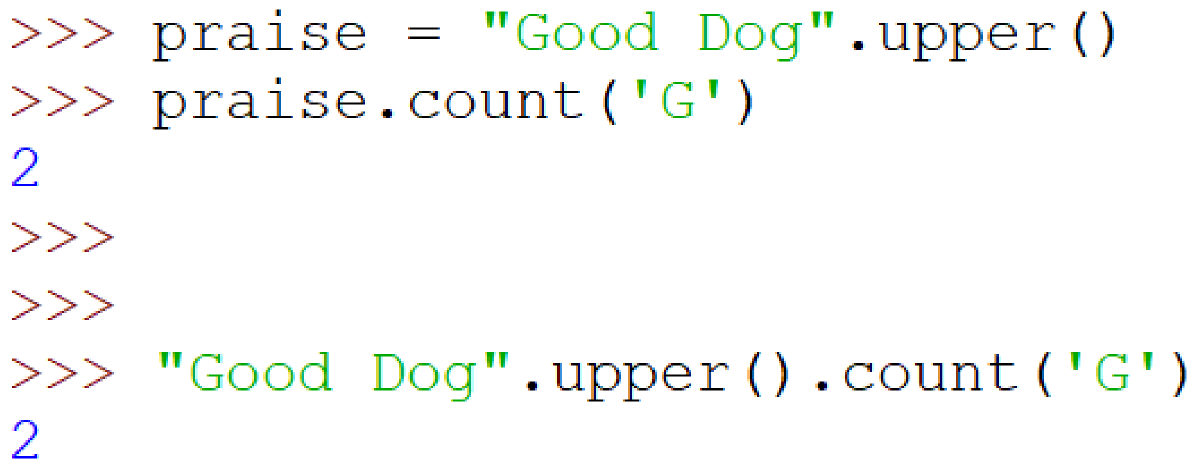
```
>>> str1=" Tai Man"
>>> str1.lstrip()  #Solution 1
'Tai Man'
>>> str1[1:] #Solution 2
'Tai Man'
```



```
>>> str2="         Tai Man"
>>> str2.lstrip()  #Solution 1 works for multiple spaces
'Tai Man'
```

Chained Methods

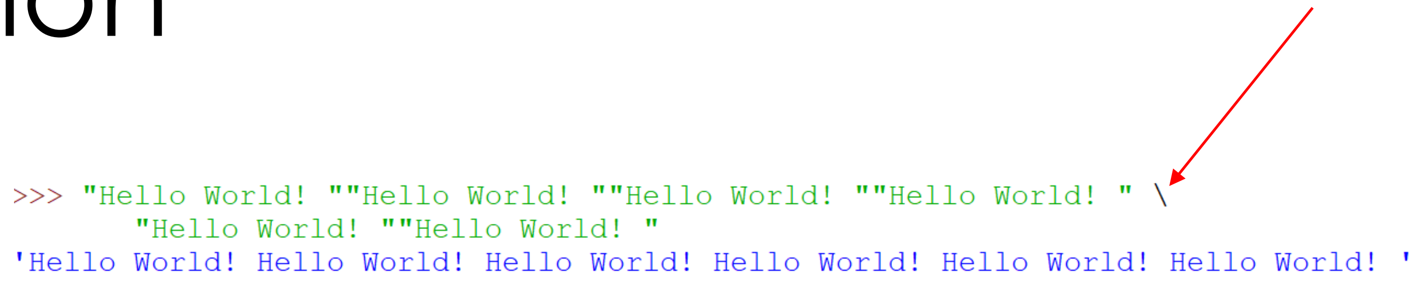
- Lines can be combined into a single line said to *chain* the two methods
    - Executed from left to right



```
>>> praise = "Good Dog".upper()
>>> praise.count('G')
2
>>>
>>>
>>> "Good Dog".upper().count('G')
2
```

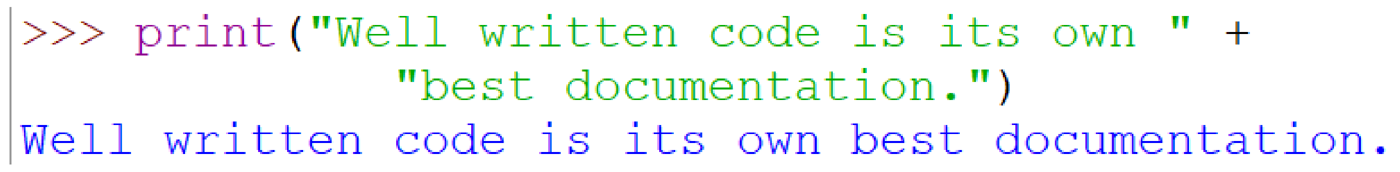
Line Continuation

- A long statement can be split across two or more lines
  - End each line with backslash character (\ )



```
>>> "Hello World! ""Hello World! ""Hello World! ""Hello World! " \
       "Hello World! ""Hello World! "
'Hello World! Hello World! Hello World! Hello World! Hello World! Hello World! '
```

- Alternatively, any code enclosed in a pair of parentheses can span multiple lines.
  - This is preferred style for most Python programmers



```
>>> print("Well written code is its own " +
                "best documentation.")
Well written code is its own best documentation.
```

Output

Section 3

Chapter 2

Optional print Argument **`sep`**

- Consider statement
  `print(value0, value1, …, valueN)`

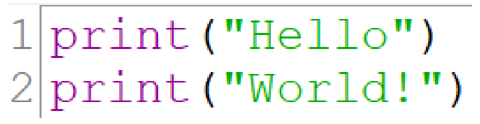
- Print function uses string consisting of *one space* character as the default separator

- One can change the separator using the **`sep`** argument

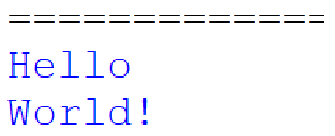
```
>>> print("a","b")
a b
>>> print("a","b", sep="&")
a&b
>>> print("a","b", sep="!!!")
a!!!b
>>> print("a","b","c", sep=" %% ")
a %% b %% c
```

Optional print Argument **`end`**

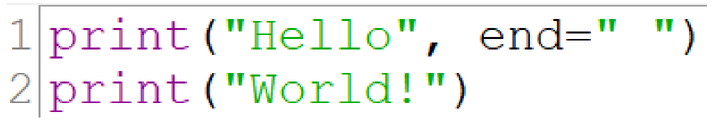
- Print statement ends by executing a newline operation.
- Once can change the ending operation with the **`end`** argument.



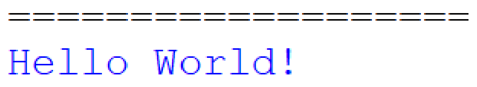
```python
print("Hello")
print("World!")
```



```
=============
Hello
World!
```



```python
print("Hello", end=" ")
print("World!")
```

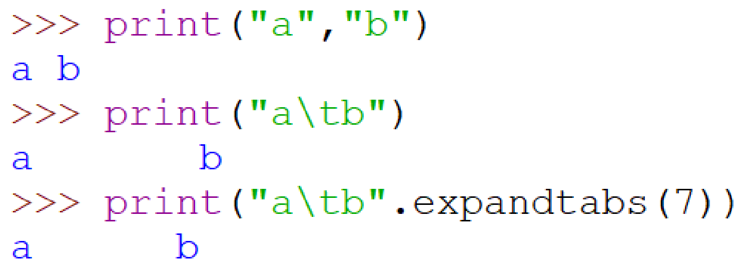


```
===================
Hello World!
```

Escape Sequences: \t

- *Escape sequences* are short sequences placed in strings
  - Instruct cursor or permit some special characters to be printed.
  - First character is always a backslash (\).

- \t   induces a horizontal tab
  - By default, the tab size is *eight* spaces
  - One can control the tab space with “expandtabs”.



```
>>> print("a","b")
a b
>>> print("a\tb")
a       b
>>> print("a\tb".expandtabs(7))
a      b
```

Escape Sequences: \n

- \n   induces a newline operation

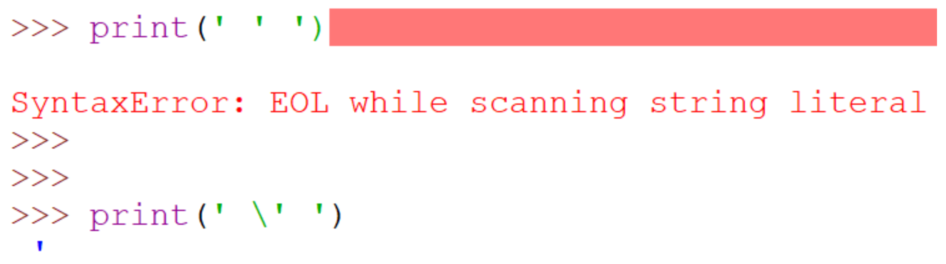
```
>>> print("ab")
ab
>>> print("a\nb")
a
b
>>> print("a\n\nb")
a

b
```

- Each escape sequence is treated as a single character.

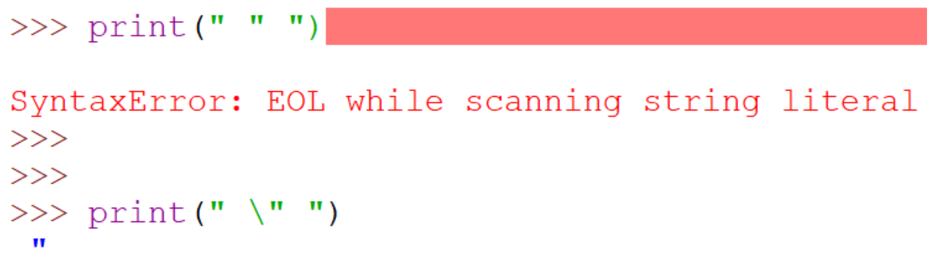
```
>>> len("a\nb")
3
>>> len("a\tb")
3
```

Backslash

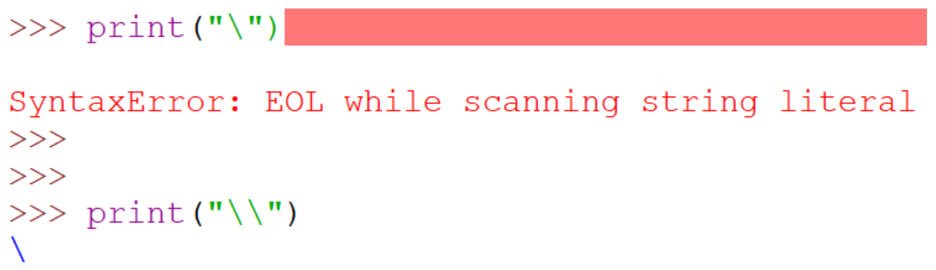
- Backslash can be used to print special characters.
  - \' causes print function to display single quotation mark
  - \" causes print function to display double quotation mark
  - \\ causes print function to display single backslash



```
>>> print(' ' ')

SyntaxError: EOL while scanning string literal
>>>
>>>
>>> print(' \' ')
 '
```



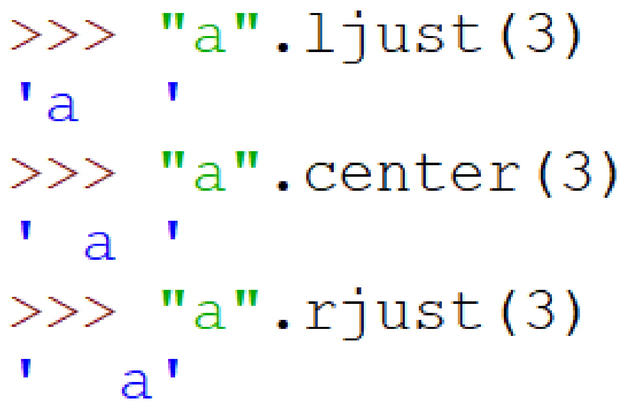
```
>>> print(" " ")

SyntaxError: EOL while scanning string literal
>>>
>>>
>>> print(" \" ")
 "
```



```
>>> print("\")

SyntaxError: EOL while scanning string literal
>>>
>>>
>>> print("\\")
\
```

Justifying Output in a Field

- Example 2: Program demonstrates methods **`ljust(w)`**, **`rjust(w)`**, and **`center(w)`**
  - **`w`** is the number of columns



```
>>> "a".ljust(3)
'a  '
>>> "a".center(3)
' a '
>>> "a".rjust(3)
'  a'
```

Justify Output with `format`

- Given: *str1* is a string and w is a field width

```
print("{0:<ws}".format(str1))
print("{0:^ws}".format(str1))
print("{0:>ws}".format(str1))
```

```
print(str1.ljust(w))
print(str1.center(w))
print(str1.rjust(w))
```

```
>>> "{0:<3s}".format("a")
'a  '
>>> "{0:^3s}".format("a")
' a '
>>> "{0:>3s}".format("a")
'  a'
```

```
>>> "a".ljust(3)
'a  '
>>> "a".center(3)
' a '
>>> "a".rjust(3)
'  a'
```

Justify Output with **`format`**

- Given: *num* is a number and *w* is a field width. `n` stands for number.

```
print("{0:<wn}".format(num))
print("{0:^wn}".format(num))
print("{0:>wn}".format(num))
```

⟺

```
print(str(num).ljust(w))
print(str(num).center(w))
print(str(num).rjust(w))
```

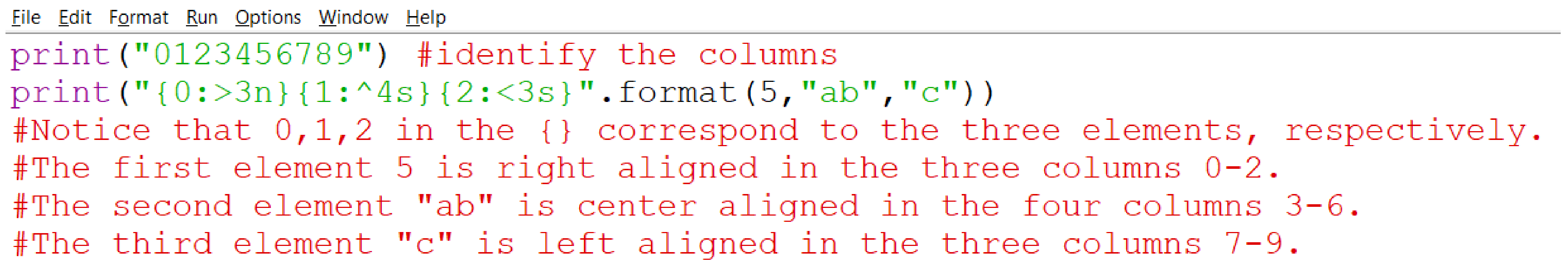
No need to convert to string

Converting to string first

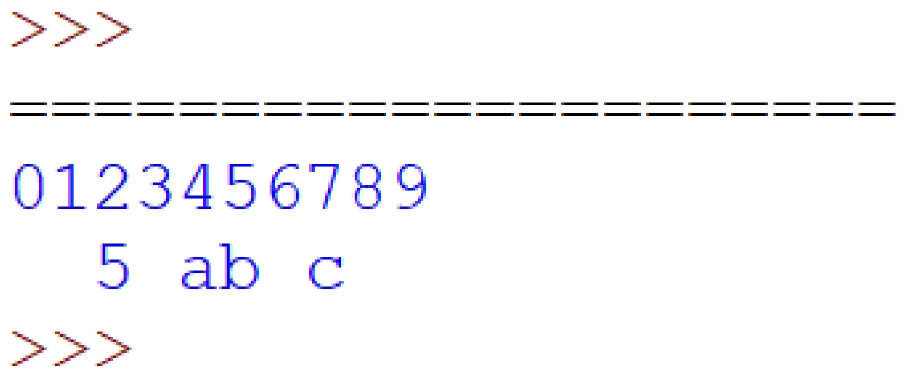
```
>>> "{0:<3n}".format(4)
'4  '
>>> "{0:^3n}".format(4)
' 4 '
>>> "{0:>3n}".format(4)
'  4'
```

```
>>> str(4).ljust(3)
'4  '
>>> str(4).center(3)
' 4 '
>>> str(4).rjust(3)
'  4'

>>> 4.ljust(3)
SyntaxError: invalid syntax
```

Justify Multiple Outputs with **`format`**



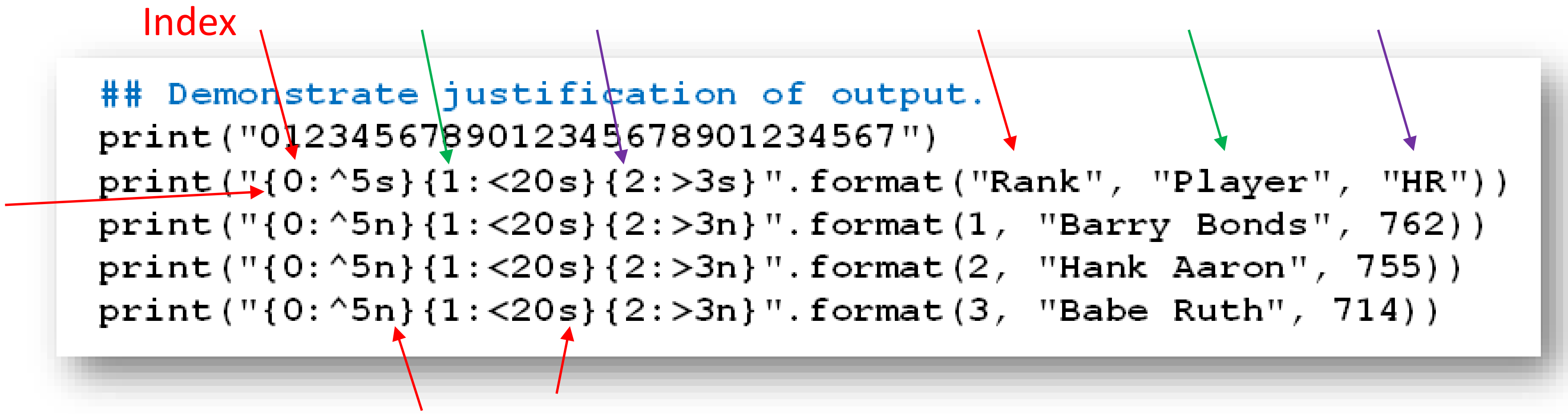
```
print("0123456789") #identify the columns
print("{0:>3n}{1:^4s}{2:<3s}".format(5,"ab","c"))
#Notice that 0,1,2 in the {} correspond to the three elements, respectively.
#The first element 5 is right aligned in the three columns 0-2.
#The second element "ab" is center aligned in the four columns 3-6.
#The third element "c" is left aligned in the three columns 7-9.
```



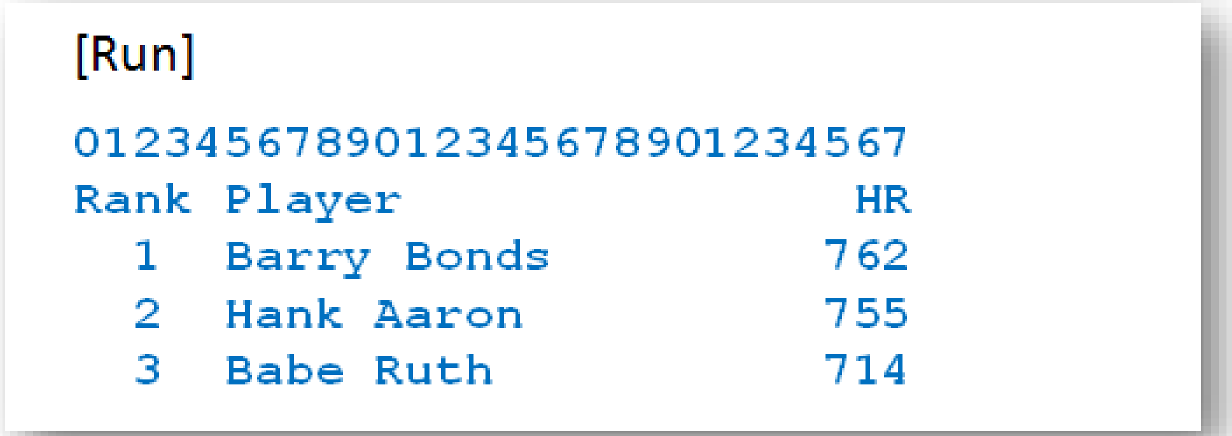
```
>>>
=====================
0123456789
  5 ab c
>>>
```

Justify Output with **`format`**

- Example 3: Program illustrates formatting



```
## Demonstrate justification of output.
print("0123456789012345678901234567")
print("{0:^5s}{1:<20s}{2:>3s}".format("Rank", "Player", "HR"))
print("{0:^5n}{1:<20s}{2:>3n}".format(1, "Barry Bonds", 762))
print("{0:^5n}{1:<20s}{2:>3n}".format(2, "Hank Aaron", 755))
print("{0:^5n}{1:<20s}{2:>3n}".format(3, "Babe Ruth", 714))
```

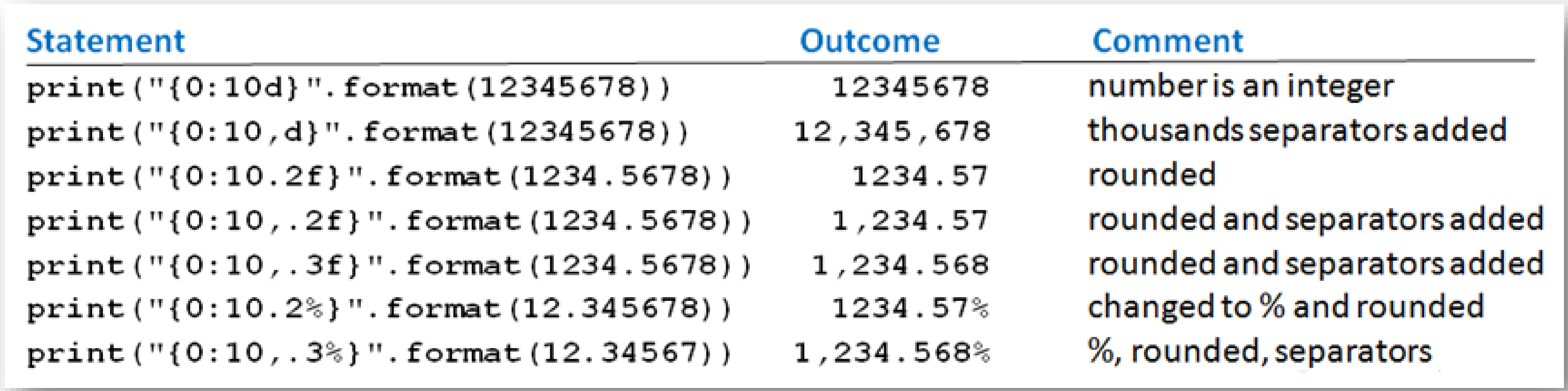


[Run]

```
0123456789012345678901234567
Rank Player               HR
  1  Barry Bonds         762
  2  Hank Aaron          755
  3  Babe Ruth           714
```

Justify Output with **`format`**

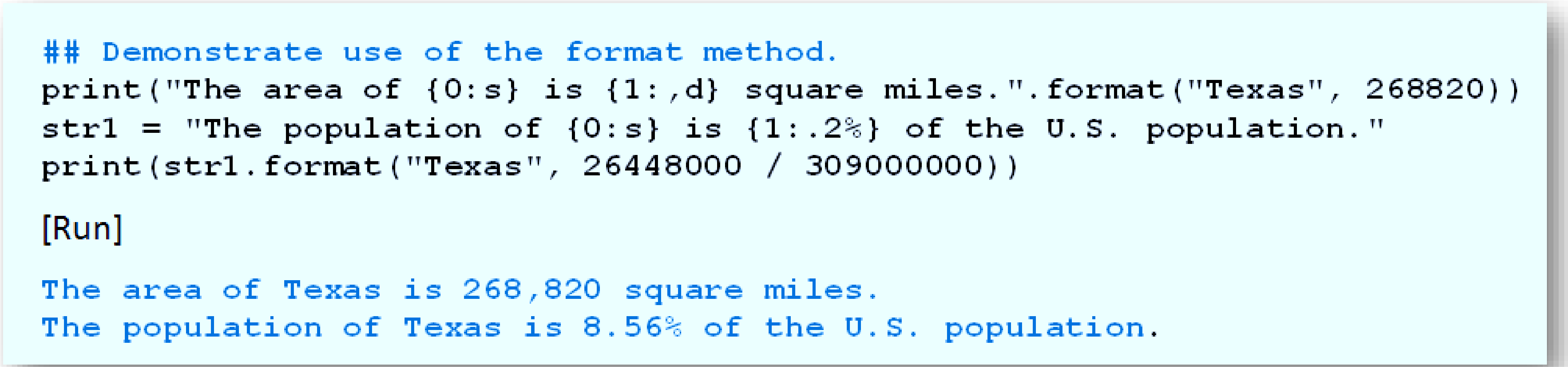
- Table 2.4 Demonstrate number formatting.



Statement	Outcome	Comment
`print("{0:10d}".format(12345678))`	`12345678`	number is an integer
`print("{0:10,d}".format(12345678))`	`12,345,678`	thousands separators added
`print("{0:10.2f}".format(1234.5678))`	`1234.57`	rounded
`print("{0:10,.2f}".format(1234.5678))`	`1,234.57`	rounded and separators added
`print("{0:10,.3f}".format(1234.5678))`	`1,234.568`	rounded and separators added
`print("{0:10.2%}".format(12.345678))`	`1234.57%`	changed to % and rounded
`print("{0:10,.3%}".format(12.34567))`	`1,234.568%`	%, rounded, separators

Justify Output with **`format`**

- Example 4: Program formatting with curly brackets



```python
## Demonstrate use of the format method.
print("The area of {0:s} is {1:,d} square miles.".format("Texas", 268820))
str1 = "The population of {0:s} is {1:.2%} of the U.S. population."
print(str1.format("Texas", 26448000 / 309000000))
```

[Run]

```
The area of Texas is 268,820 square miles.
The population of Texas is 8.56% of the U.S. population.
```

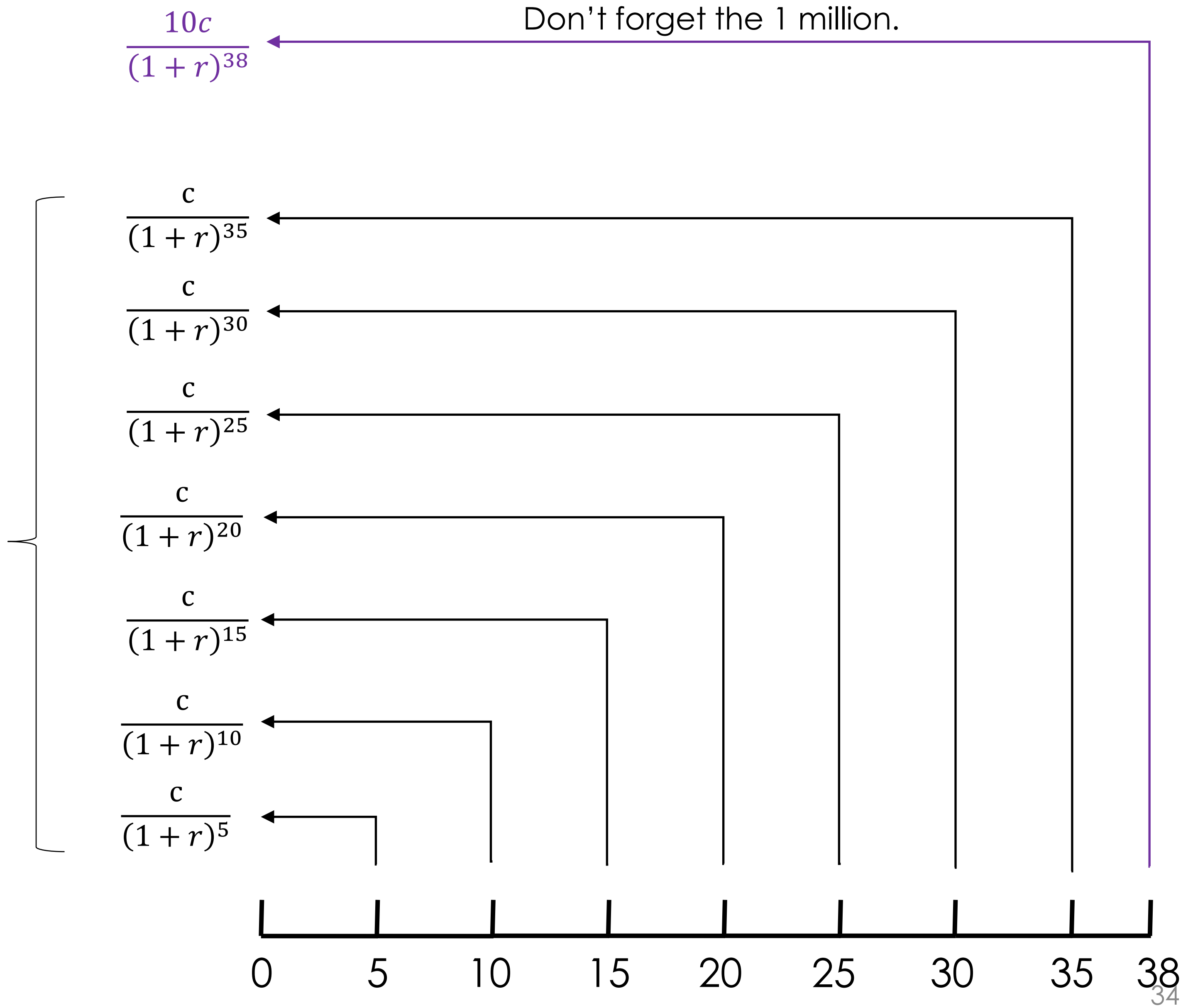
Lab 3

Homework 2

Let r be the annual rate, $\mathrm{c} = 100{,}000$.

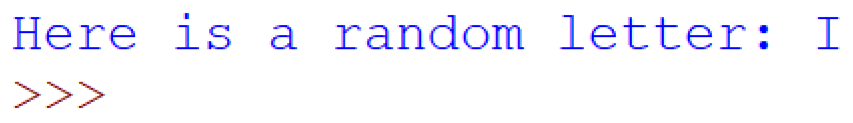
To calculate the PV of an annuity, we essentially apply the PV $= \frac{\text{FV}}{(1+r)^m}$ formula repeatedly with different m's to get the PV of each future cash flow. Then we add them up.

Three ways to do the summation:
1) Use the formula in the original hint.
2) By brutal force (term by term).
3) Use a loop.

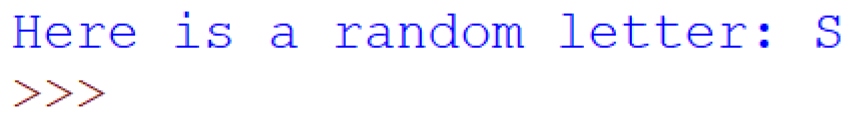


$\frac{10c}{(1+r)^{38}}$
Don't forget the 1 million.
$\frac{\mathrm{c}}{(1+r)^{35}}$
$\frac{\mathrm{c}}{(1+r)^{30}}$
$\frac{\mathrm{c}}{(1+r)^{25}}$
$\frac{\mathrm{c}}{(1+r)^{20}}$
$\frac{\mathrm{c}}{(1+r)^{15}}$
$\frac{\mathrm{c}}{(1+r)^{10}}$
$\frac{\mathrm{c}}{(1+r)^{5}}$
0
5
10
15
20
25
30
35
38

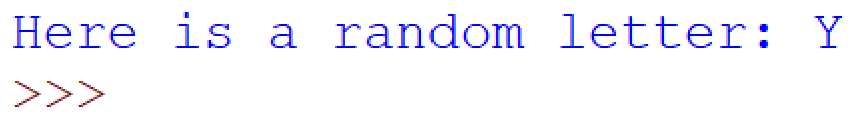
Example A: generate a random letter



```
Here is a random letter: I
>>>
```

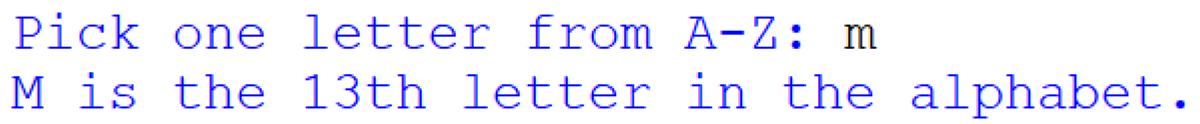


```
Here is a random letter: S
>>>
```



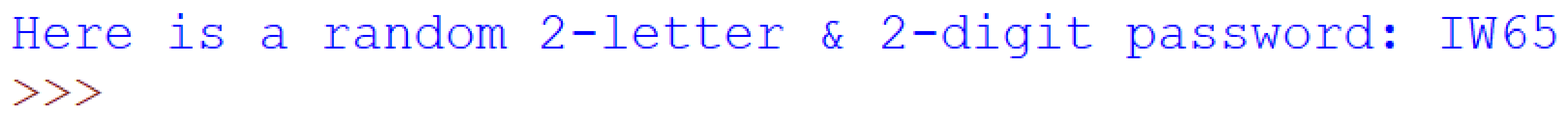
```
Here is a random letter: Y
>>>
```

Example B: alphabetic order

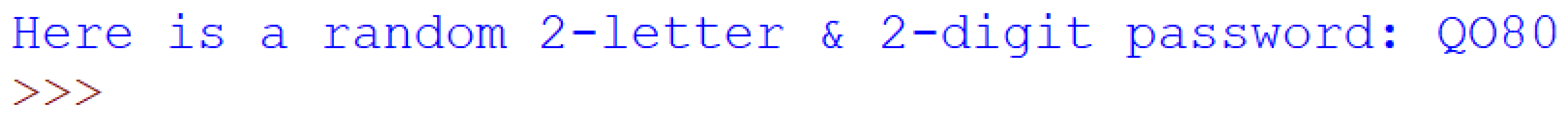


```
Pick one letter from A-Z: m
M is the 13th letter in the alphabet.
```

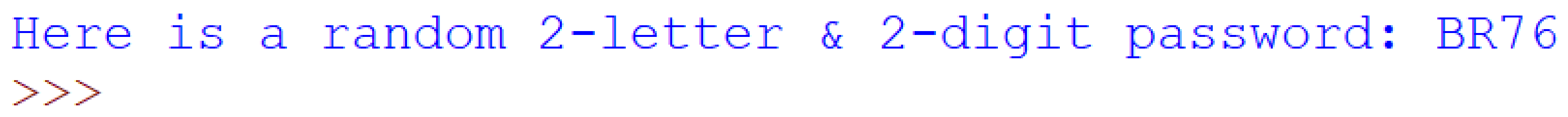
Example C: random password



```
Here is a random 2-letter & 2-digit password: IW65
>>>
```



```
Here is a random 2-letter & 2-digit password: QO80
>>>
```



```
Here is a random 2-letter & 2-digit password: BR76
>>>
```

Classwork_3

- Write a Python program for word replacement in a sentence:
  - Ask the user to input a sentence, a word in the sentence, and a replacement word.
  - Display the sentence with the original word replaced.
  - The output should resemble the following. Include the exit line. Upload the .py file on Canvas.
- Hint: The following uses the screen captured sentence as an example. The program needs to work for general user inputs.
  - Step 1: Take out "complaint": slice the sentence string into two pieces: "No " and ", no gain".
  - Step 2: Insert "pain" between the two pieces: concatenate the three strings in the correct order.