COMP2853

LESSON 4: STRINGS

Agenda – Lesson 4

- Quick Review
  - Lab 3
  - Branching and While Loops
  - Pre-Reading - Strings
- Quiz 3
  - Review Answers
- Assignment 1 Preview
- Strings
- Lab 4 – In Class
- Homework – Lab 4 Take Home and Pre-Reading

Quiz 3

- This is an individual assessment, please do your own work

- You have 20 minutes to complete it.

- We will go over the answers afterwards

Course Outline

#	Topics	Quiz	Lab	Assignment
1	Python Setup, Variables, Expressions, Naming Conventions		Lab 1	
2	Logical Operators, Functions and Modules, Main Function	Quiz 1	Lab 2	
3	Branching and Loops	Quiz 2	Lab 3	
4	Strings	Quiz 3	Lab 4	
5	Lists, Tuples, Dictionaries, and Sets	Quiz 4	Lab 5	
6	Pandas	Midterm	Lab 6	Assignment 1 Due

Command Line Arguments (Preview)

AKA Comand Line Parameters

- Sometimes you want to pass arguments to the Python program itself

- Example
  - main.py Susan Smith
  - Susan is the first name
  - Smith is the last name

- Python places the arguments into a list called sys.argv
  - Element 0: The program name (main.py)
  - Element 1: The first argument
  - Element 2: The second argument
  - And so on

- Need to import sys (built-in Python module) to access sys.argv

Script main.py

```
import sys


def main():
    if len(sys.argv) <= 2:
        print("Not enough command line arguments")
        exit(0)
    elif len(sys.argv) > 3:
        print("Too many command line arguments")
        exit(0)


    print("Hello, %s, %s" % (sys.argv[1], sys.argv[2]))


if __name__ == "__main__":
    main()
```

Need to import sys to get the arguments

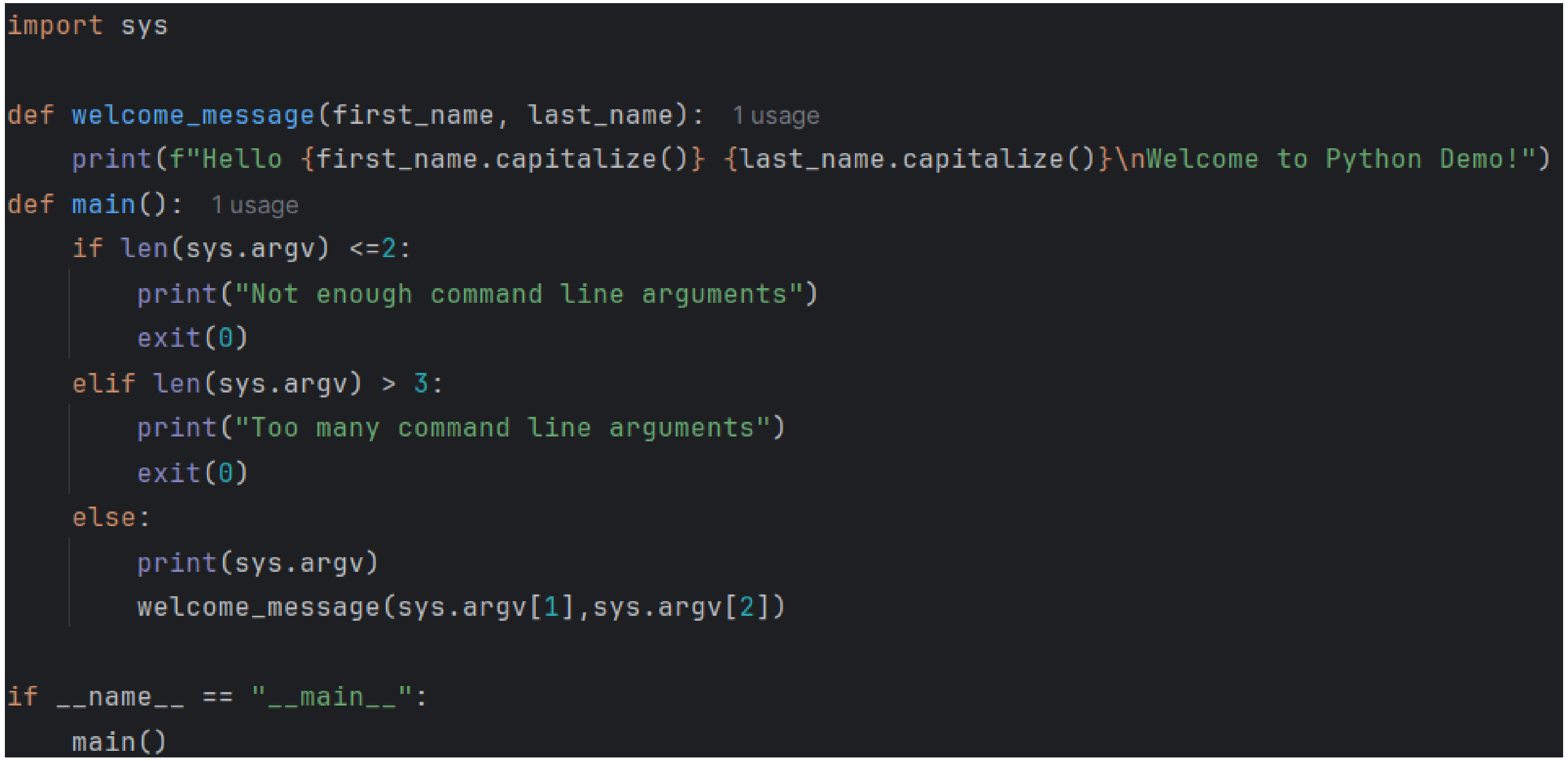
sys.argv is a list of strings

You can use **len** to get the length of a list

We want to make sure we have the correct number of arguments

sys.argv[0] – the name of the script
sys.argv[1] – the first argument
sys.argv[2] – the second argument
And so on…

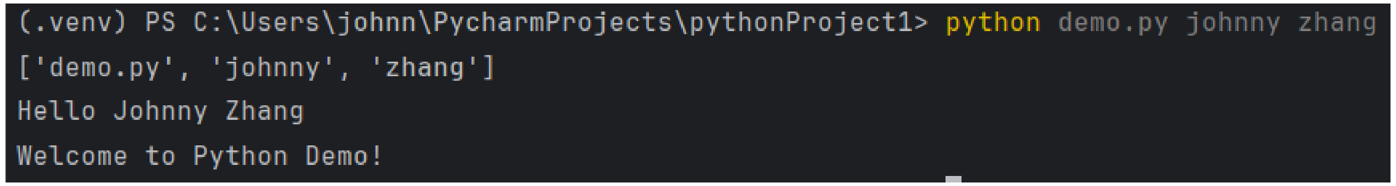
Demo



```
import sys

def welcome_message(first_name, last_name):  1 usage
    print(f"Hello {first_name.capitalize()} {last_name.capitalize()}\nWelcome to Python Demo!")
def main():  1 usage
    if len(sys.argv) <=2:
        print("Not enough command line arguments")
        exit(0)
    elif len(sys.argv) > 3:
        print("Too many command line arguments")
        exit(0)
    else:
        print(sys.argv)
        welcome_message(sys.argv[1],sys.argv[2])

if __name__ == "__main__":
    main()
```



```
(.venv) PS C:\Users\johnn\PycharmProjects\pythonProject1> python demo.py johnny zhang
['demo.py', 'johnny', 'zhang']
Hello Johnny Zhang
Welcome to Python Demo!
```

Learning Outcomes:

- Strings
- Special characters in a string
- String operators
- Formatting a string
- Data format : CSV
- Data format: JSON
- String indexing and slicing
- String methods

String

- A string is a sequence of zero or more alphanumeric characters or symbols enclosed in quotation marks (single or double)
- Example:

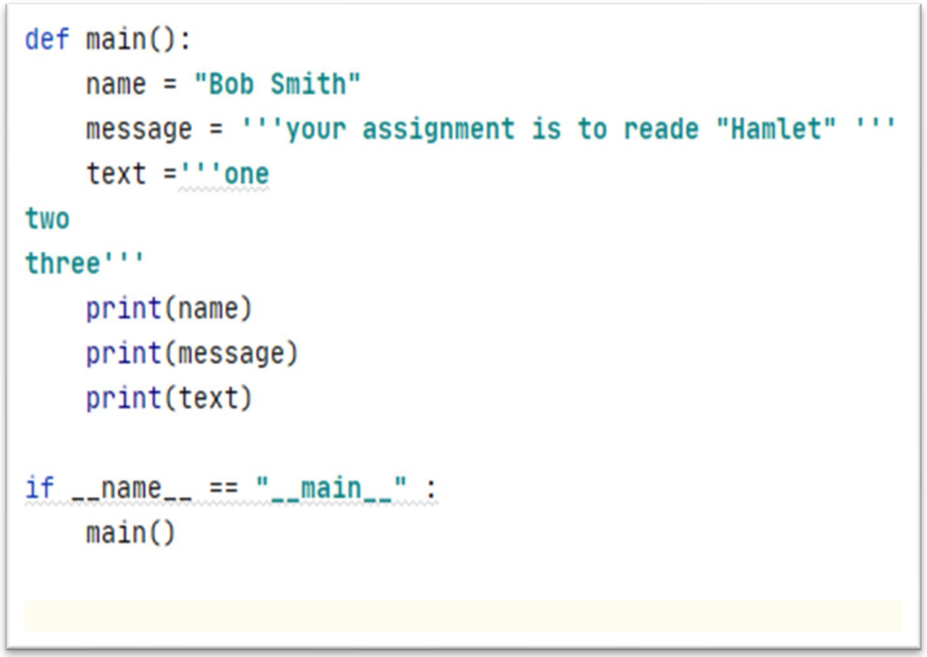
  “Hello World”

  full_name = “John Doe”
- full_name is a variable of type string
- “John Doe” is a string literal (string value)

String

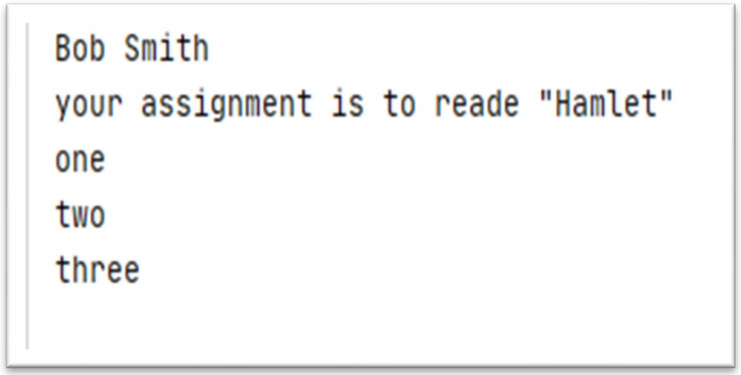
- If the string contains single quotes, use double quotes to enclose the string
- If the string literal contains single or double quotes use triple quotes(“”” or ‘”) to enclose the string
- **Example:**

Output:



```
def main():
    name = "Bob Smith"
    message = '''your assignment is to reade "Hamlet" '''
    text ='''one
two
three'''
    print(name)
    print(message)
    print(text)

if __name__ == "__main__" :
    main()
```



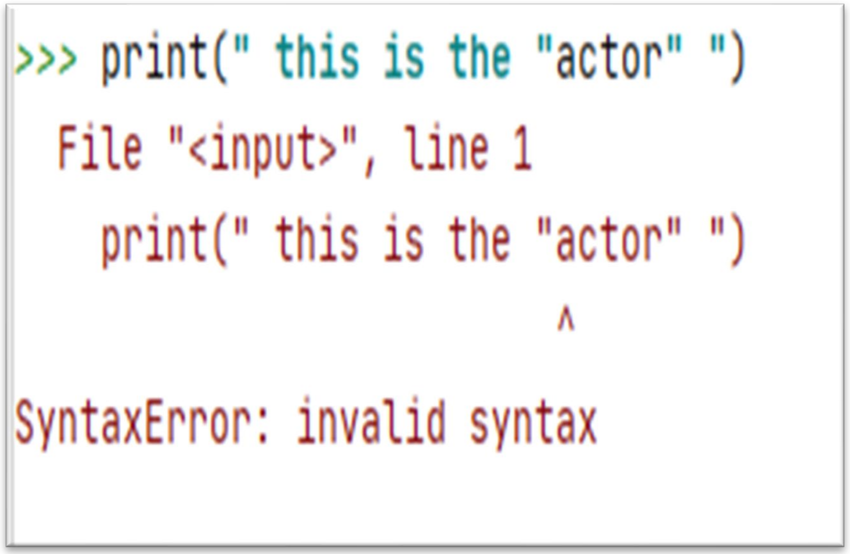
```
Bob Smith
your assignment is to reade "Hamlet"
one
two
three
```

String : special characters

- The special characters are symbols that have special, built-in meaning in a programming language.
- Example :
- In Python: ',",
- In python the backslash (\) character is used to escape characters that otherwise have special meaning.

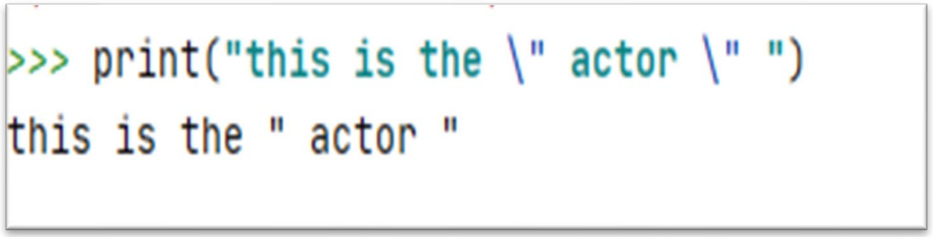
String: Escape Character

A string using a special character without the backslash.



```
>>> print(" this is the "actor" ")
  File "<input>", line 1
    print(" this is the "actor" ")
                         ^
SyntaxError: invalid syntax
```

A string using a special character with the backslash.

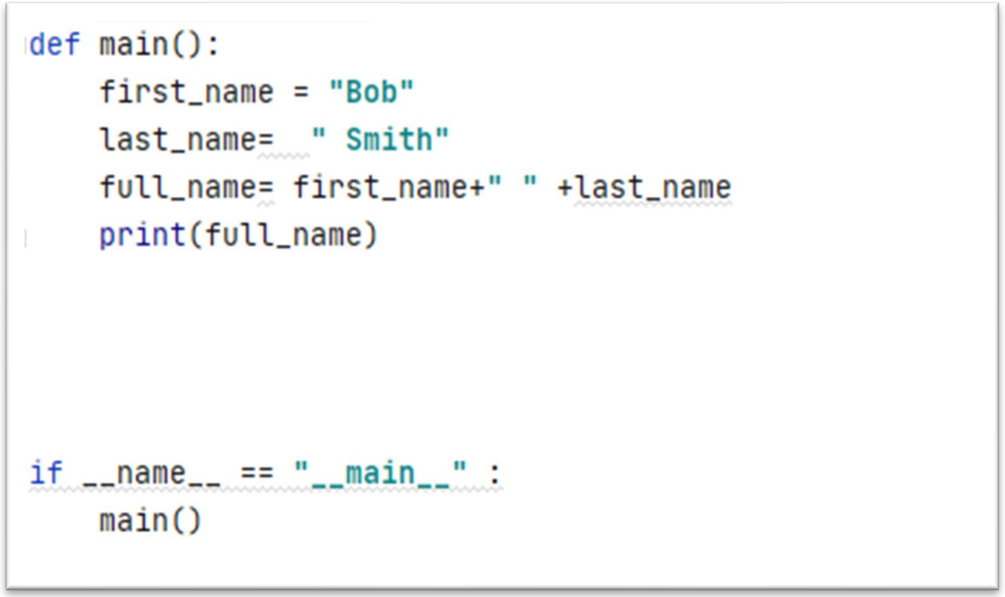


```
>>> print("this is the \" actor \" ")
this is the " actor "
```

String: String Operators

- + is used to join (concatenate Strings together).

Example:

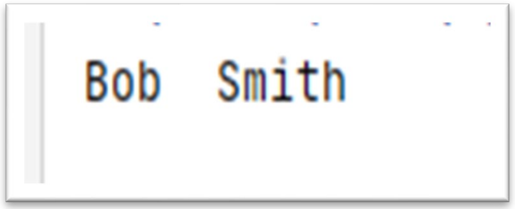


```
def main():
    first_name = "Bob"
    last_name=  " Smith"
    full_name= first_name+" " +last_name
    print(full_name)




if __name__ == "__main__" :
    main()
```

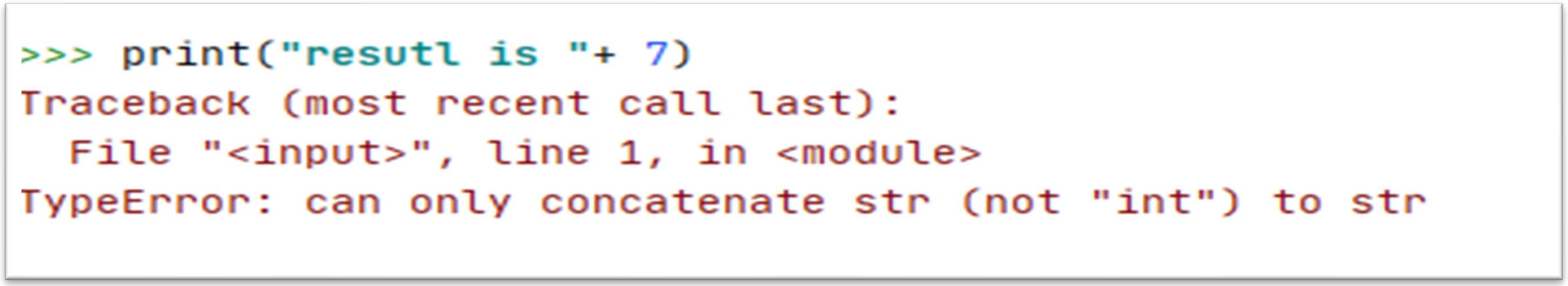
Output:



```
Bob  Smith
```

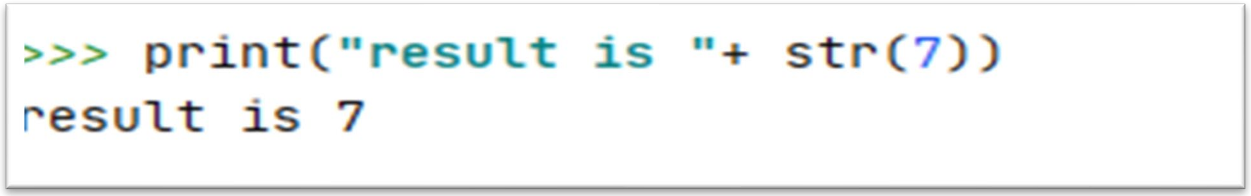
String: String Operators

- Any other type such as int **will not** be converted to a String with +
- Example:



```
>>> print("resutl is "+ 7)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

- Use function str() to convert any type to string
- Example:

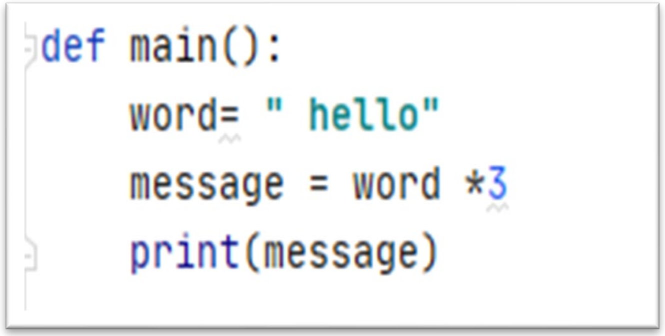


```
>>> print("result is "+ str(7))
result is 7
```

String: String Operators

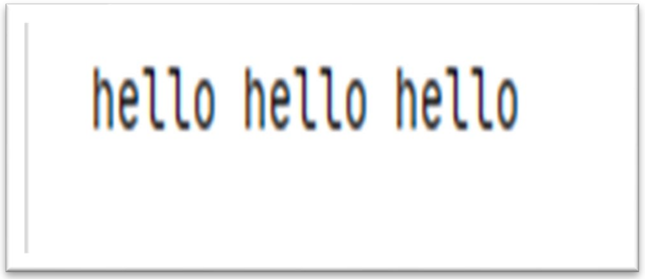
- Symbol Asterisk * is used to repeat the String

Example:



```
def main():
    word= " hello"
    message = word *3
    print(message)
```

Output:



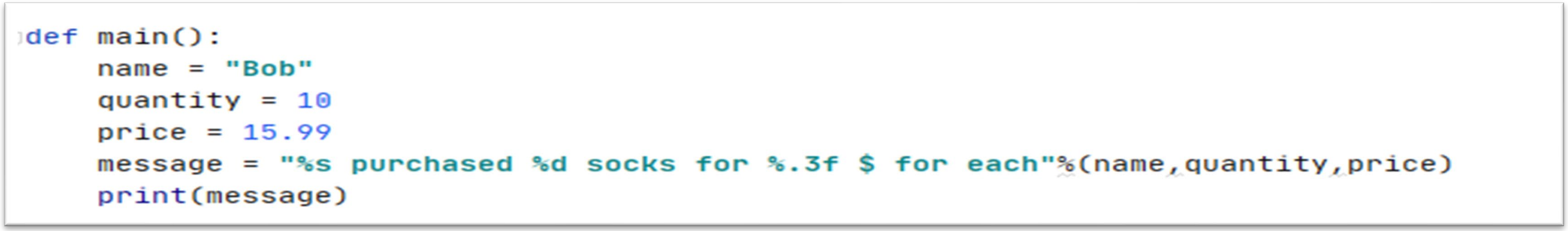
```
hello hello hello
```

- What will be the result of executing the following statement?
  print(“ba” + “na” * 2)

Formatting a String

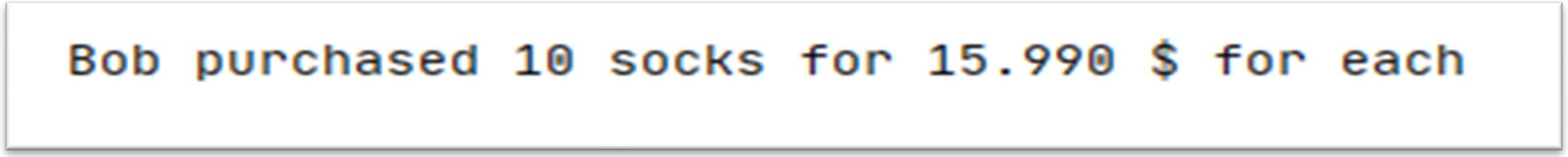
- String formatting expression is a String that is created with placeholders that are replaced by the values of variables.

Example:



```
def main():
    name = "Bob"
    quantity = 10
    price = 15.99
    message = "%s purchased %d socks for %.3f $ for each"%(name,quantity,price)
    print(message)
```

Output:



```
Bob purchased 10 socks for 15.990 $ for each
```

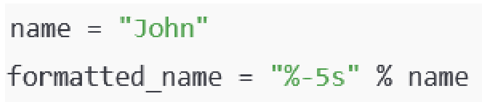
Formatting a String

- Some of the used specifiers in String formatting expression

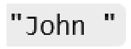
%s	- String (or any object with a string representation, like numbers)
%d	- Integer
%f	- Floating point number
%.< number of digits>f	- Floating point numbers with a fixed amount of digits to the right of the dot.
%x/%X	- Integers in hex representation (lowercase/uppercase)
%e/%E	- Floating points in Exponential format (lowercase/uppercase (Ex: 1.7e3))
%% percentage sign (%)	- This is how you escape the % sign

Formatting a String

- Minimal field width
  - %5s # minimum 5 characters reserved
  - %3d # minimum 3 digits
- Zero padding
  - %03d # adds leading zeros to fill the space. So 20 displays as 020.
- Left justification
  - %-5s # string is justified to the left rather than the right



```
name = "John"
formatted_name = "%-5s" % name
```

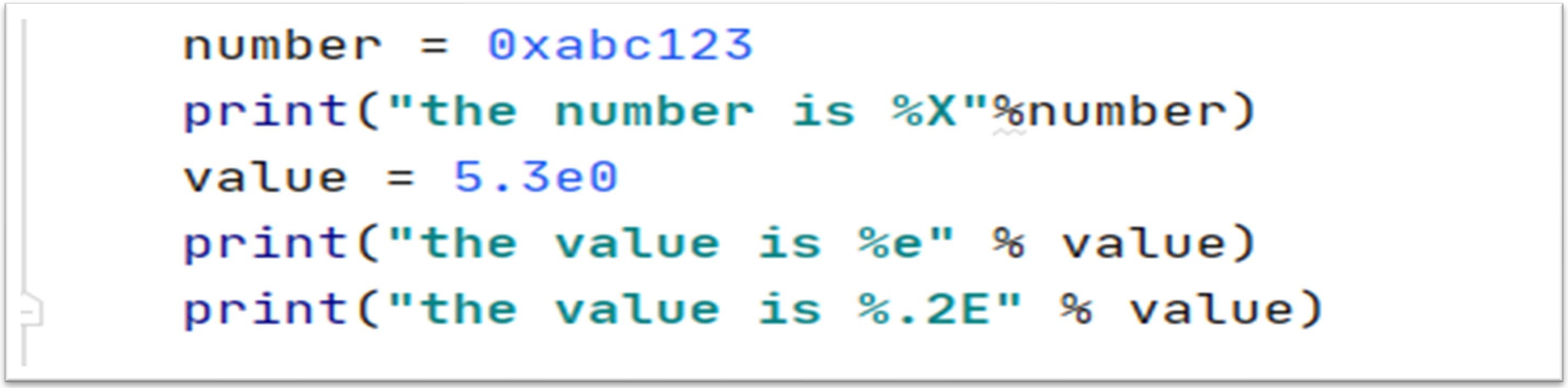


```
"John "
```

- Floating point precision
  - %.3f # Displays 3 decimal places so 1.5 displays as 1.500

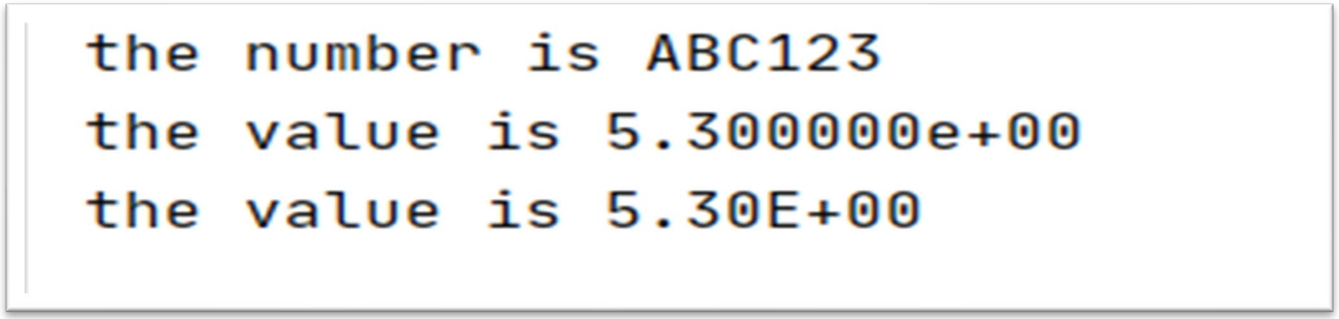
Formatting a String

Example:



```
number = 0xabc123
print("the number is %X"%number)
value = 5.3e0
print("the value is %e" % value)
print("the value is %.2E" % value)
```

Output:

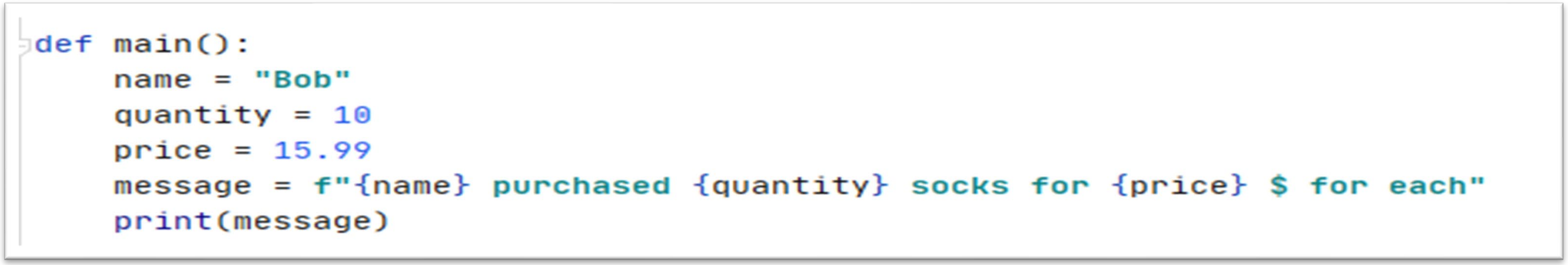


```
the number is ABC123
the value is 5.300000e+00
the value is 5.30E+00
```

Formatting a String: Using f-string

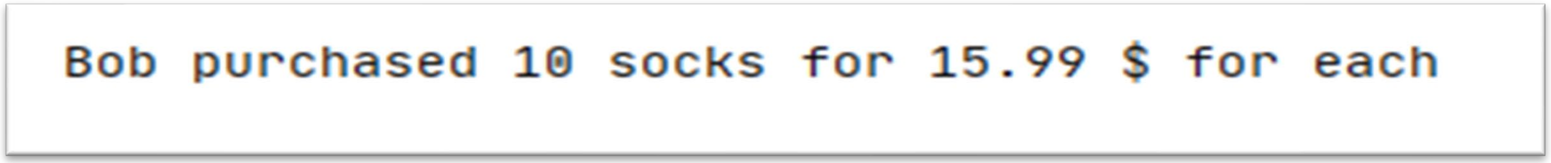
- f-string is an alternate form of string formatting expressions recently introduced to Python.
- It is specified by putting f in front of the starting quote of the string, i.e. *f "This is an f-string with a {placeholder}"*

Example:



```
def main():
    name = "Bob"
    quantity = 10
    price = 15.99
    message = f"{name} purchased {quantity} socks for {price} $ for each"
    print(message)
```

Output:

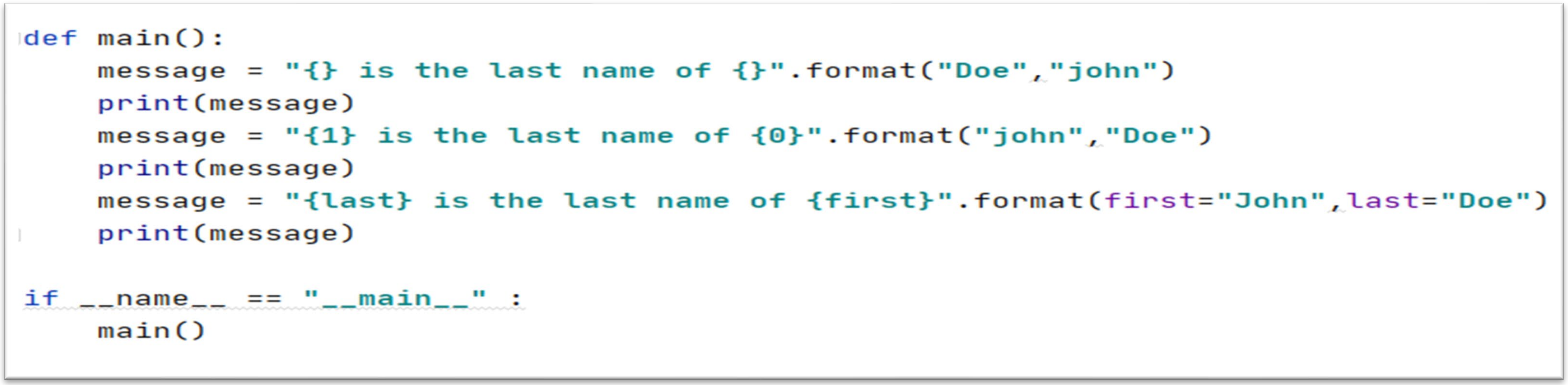


```
Bob purchased 10 socks for 15.99 $ for each
```

Formatting a String: format()

- format method is an alternative way to format a string expression.

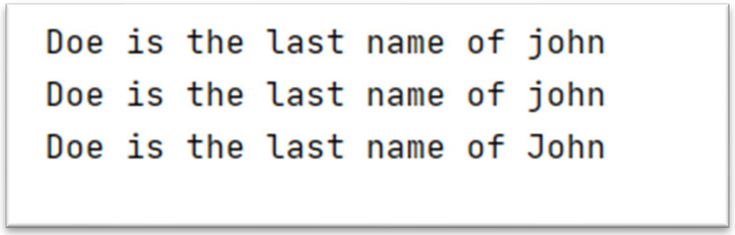
Example:



```
def main():
    message = "{} is the last name of {}".format("Doe","john")
    print(message)
    message = "{1} is the last name of {0}".format("john","Doe")
    print(message)
    message = "{last} is the last name of {first}".format(first="John",last="Doe")
    print(message)

if __name__ == "__main__" :
    main()
```

Output:



```
Doe is the last name of john
Doe is the last name of john
Doe is the last name of John
```

Data Formats: CSV

- CSV – Comma-Separated Values for tabular data.
- Each row represents a record.
- Each column represents a field.
- The comma separates the columns in the row.

Susan	Lee	18	34.58
Robert	Smith	5	25.75
Jennifer	Chen	10	30.00

- Often stored as :

```
Susan,Lee,18,34.58
Robert,Smith,5,25.75
Jennifer,Chen,10,30.00
```

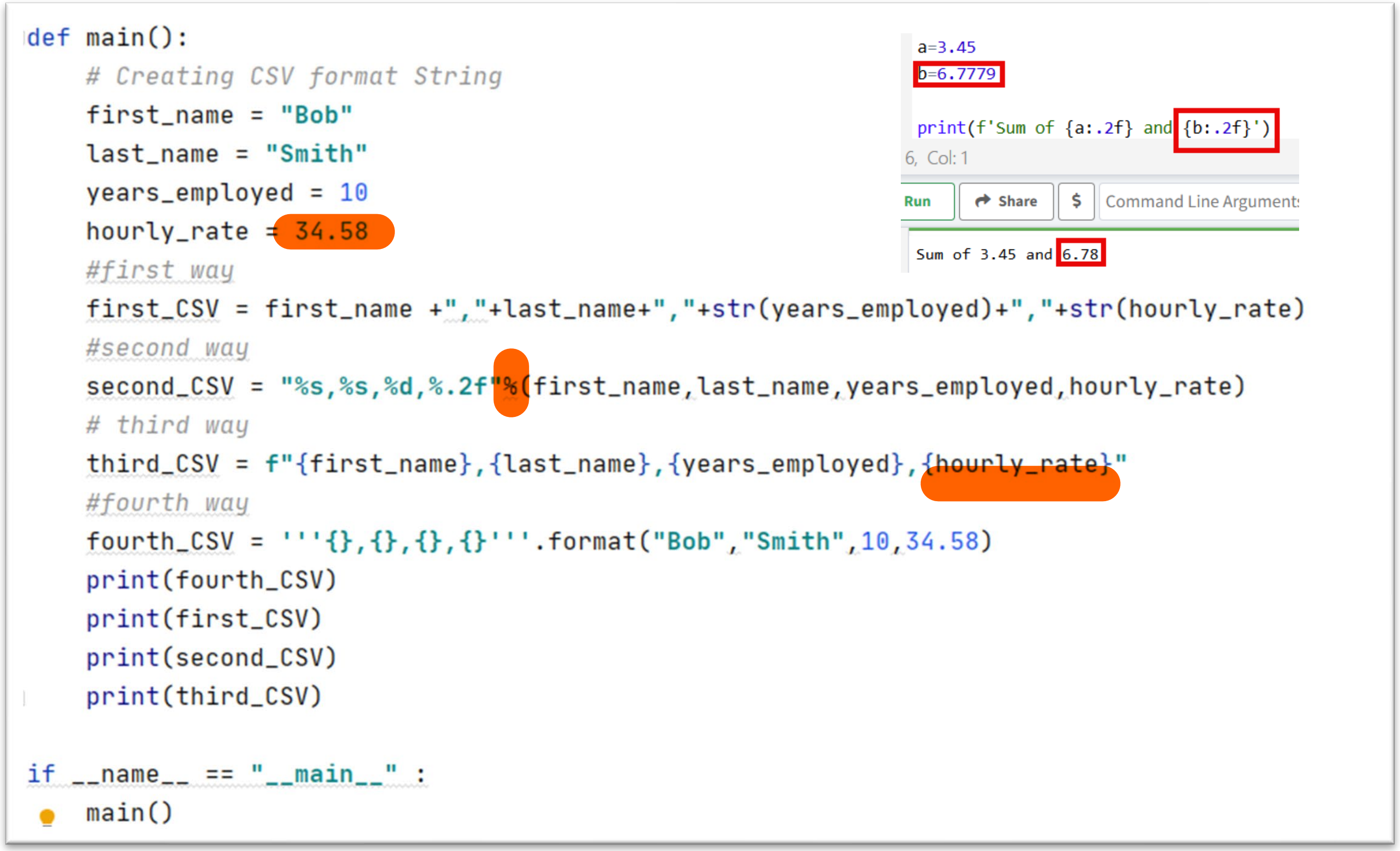
Data Formats: CSV

Given the following variables:

```
first_name = "Bob"
last_name = "Smith"
years_employed = 10
hourly_rate_cad = 34.58
```

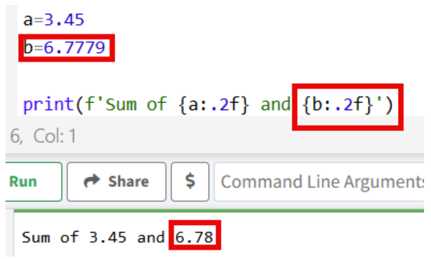
Create a string in a CSV format

Data Formats: CSV



```
def main():
    # Creating CSV format String
    first_name = "Bob"
    last_name = "Smith"
    years_employed = 10
    hourly_rate = 34.58
    #first way
    first_CSV = first_name +","+last_name+","+str(years_employed)+","+str(hourly_rate)
    #second way
    second_CSV = "%s,%s,%d,%.2f"%(first_name,last_name,years_employed,hourly_rate)
    # third way
    third_CSV = f"{first_name},{last_name},{years_employed},{hourly_rate}"
    #fourth way
    fourth_CSV = '''{},{},{},{}'''.format("Bob","Smith",10,34.58)
    print(fourth_CSV)
    print(first_CSV)
    print(second_CSV)
    print(third_CSV)

if __name__ == "__main__" :
    main()
```

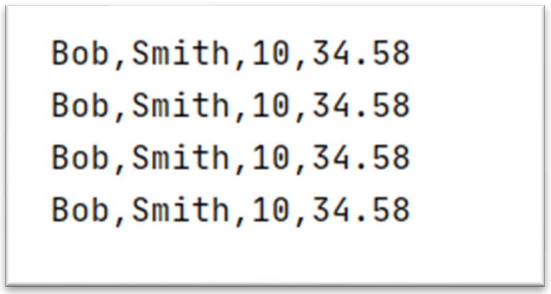


```
a=3.45
b=6.7779

print(f'Sum of {a:.2f} and {b:.2f}')
```

```
Sum of 3.45 and 6.78
```

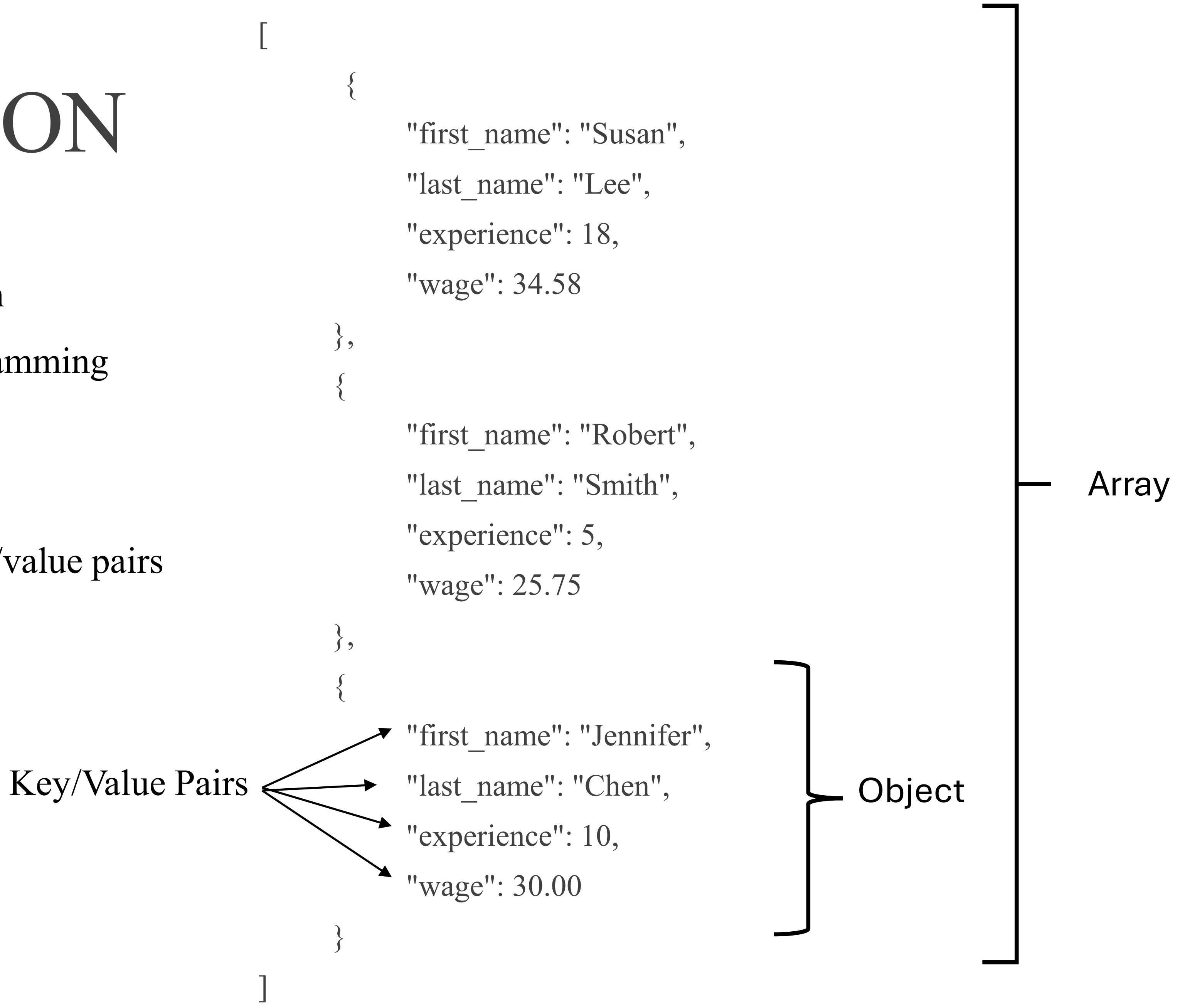
The Output:



```
Bob,Smith,10,34.58
Bob,Smith,10,34.58
Bob,Smith,10,34.58
Bob,Smith,10,34.58
```

Data Formats: JSON

- JSON – JavaScript Object Notation
- Derived from the JavaScript Programming Language
- But it is used "everywhere"
- Consists of arrays, objects and key/value pairs



```
[
    {
        "first_name": "Susan",
        "last_name": "Lee",
        "experience": 18,
        "wage": 34.58
    },
    {
        "first_name": "Robert",
        "last_name": "Smith",
        "experience": 5,
        "wage": 25.75
    },
    {
        "first_name": "Jennifer",
        "last_name": "Chen",
        "experience": 10,
        "wage": 30.00
    }
]
```

Data Formats: JSON

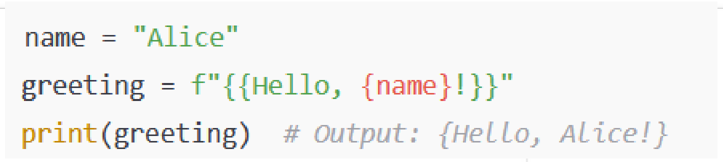
Given the following variables:

```
first_name = ”Bob”
last_name = “Smith”
years_employed = 10
hourly_rate = 34.58
```

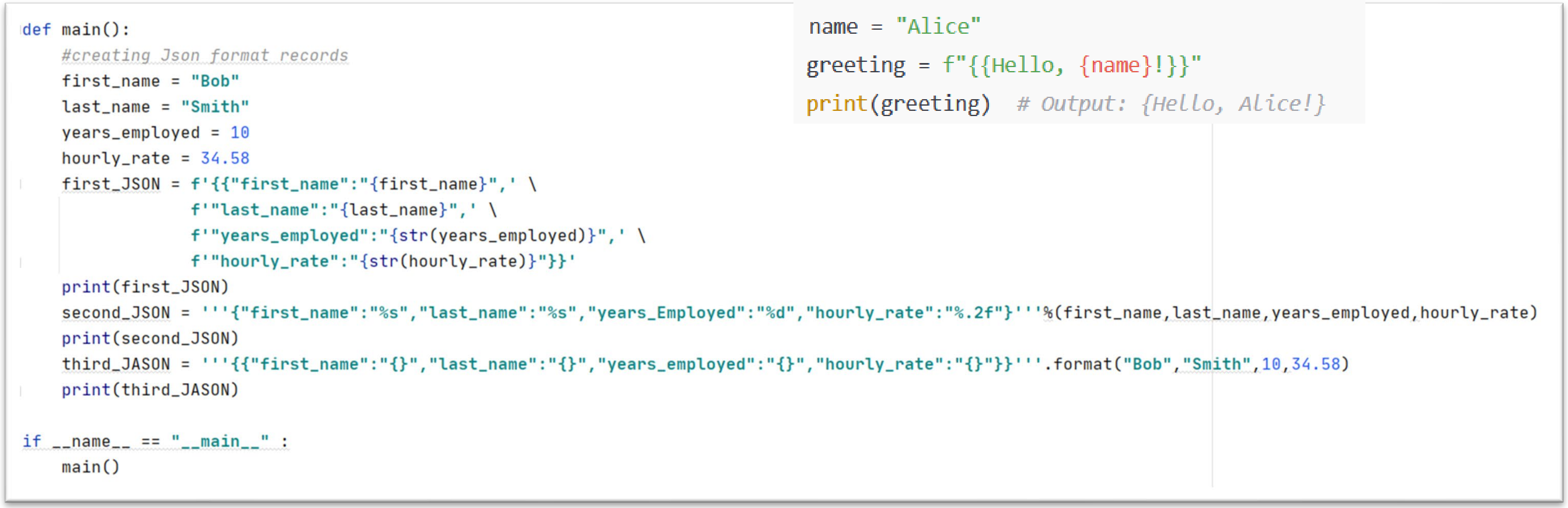
Create a string in JSON format

Data Formats: JSON

{{ and }} are used to escape the curly braces and include them in the string output.



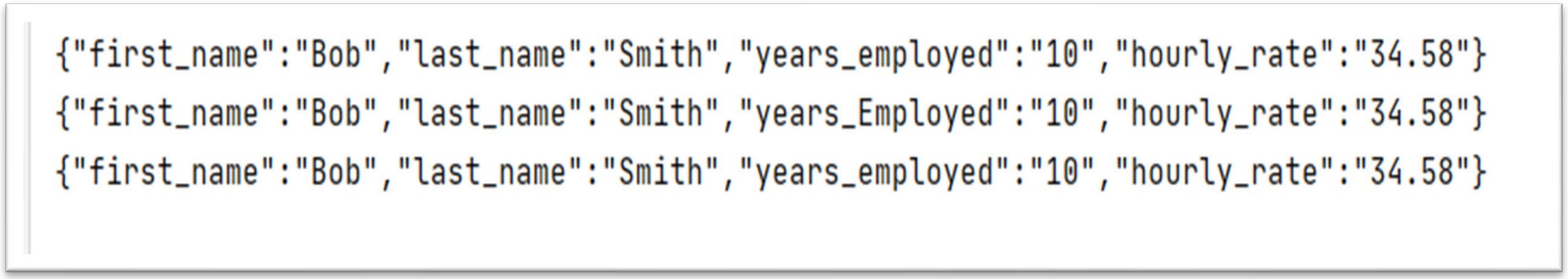
```python
name = "Alice"
greeting = f"{{Hello, {name}!}}"
print(greeting)  # Output: {Hello, Alice!}
```



```python
def main():
    #creating Json format records
    first_name = "Bob"
    last_name = "Smith"
    years_employed = 10
    hourly_rate = 34.58
    first_JSON = f'{{"first_name":"{first_name}",' \
                 f'"last_name":"{last_name}",' \
                 f'"years_employed":"{str(years_employed)}",' \
                 f'"hourly_rate":"{str(hourly_rate)}"}}'
    print(first_JSON)
    second_JSON = '''{"first_name":"%s","last_name":"%s","years_Employed":"%d","hourly_rate":"%.2f"}'''%(first_name,last_name,years_employed,hourly_rate)
    print(second_JSON)
    third_JASON = '''{{"first_name":"{}","last_name":"{}","years_employed":"{}","hourly_rate":"{}"}}'''.format("Bob","Smith",10,34.58)
    print(third_JASON)

if __name__ == "__main__" :
    main()
```

The output is:



```
{"first_name":"Bob","last_name":"Smith","years_employed":"10","hourly_rate":"34.58"}
{"first_name":"Bob","last_name":"Smith","years_Employed":"10","hourly_rate":"34.58"}
{"first_name":"Bob","last_name":"Smith","years_employed":"10","hourly_rate":"34.58"}
```

Data Formats

- We'll be using these data formats, and others, throughout the rest of the course.

- CSV is commonly used with spreadsheets.
- JSON is commonly used in web-based interfaces.

Strings: Indexing and Slicing

- Remember that the string is a sequence of characters.
- Each character has an int value to indicate its position in the string (index).
- Each character in a string can be accessed using its index.
- Indexing starts at 0 so the first character index is 0.

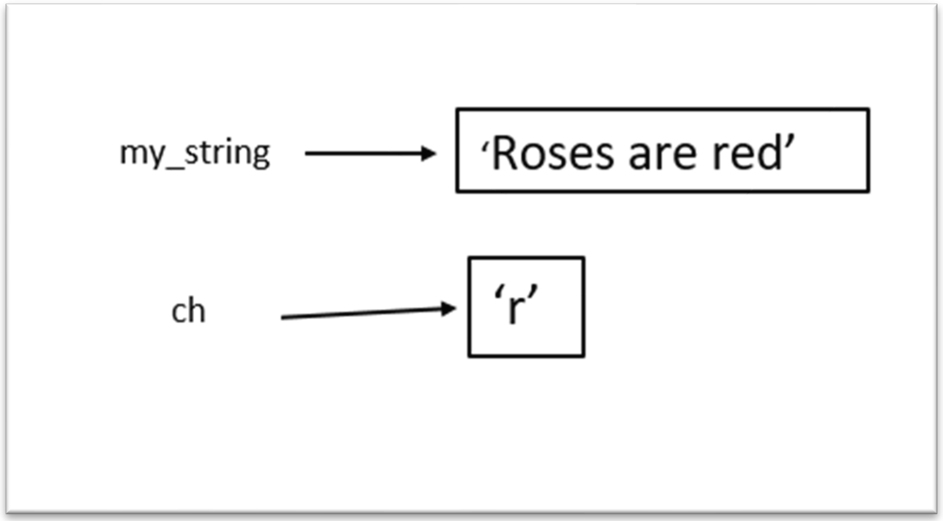
R	**o**	**s**	**e**	**s**		**a**	**r**	**e**		**r**	**e**	**d**
0	1	2	3	4	5	6	7	8	9	10	11	12

String: Indexing and Slicing

Example:

R	o	s	e	s		a	r	e		r	e	d
0	1	2	3	4	5	6	7	8	9	10	11	12

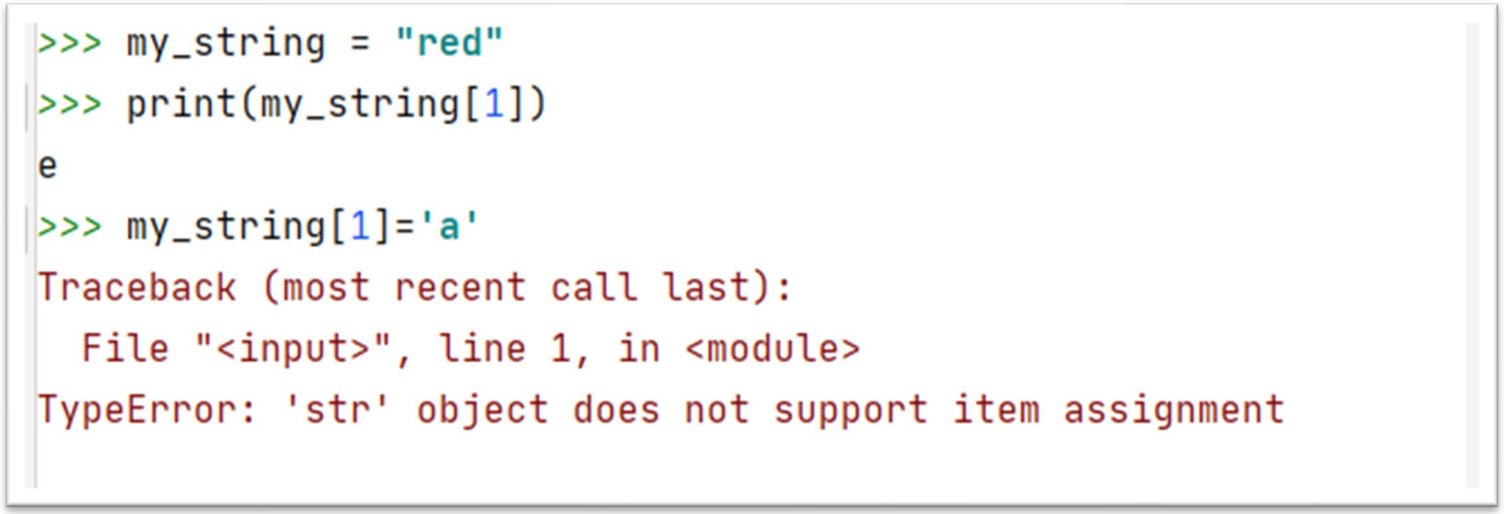
```
my_string = "Roses are red"
ch = my_string[7]
print(ch)
```



The above code will display: r

String: Indexing and Slicing

- Be careful! You can use the index to access the character but you cannot use the index to change the character.

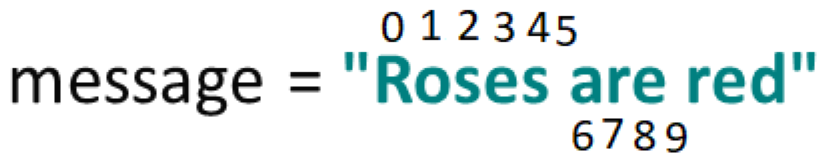


```
>>> my_string = "red"
>>> print(my_string[1])
e
>>> my_string[1]='a'
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

- This is because a string is <u>immutable</u>; once it is created it cannot be changed.

String Slicing

- A segment of a string is called a slice. Selecting a slice is similar to selecting a character.
- When slicing a string you are creating a substring from the original string.
- Example:



```
           0 1 2 3 4 5
message = "Roses are red"
                 6 7 8 9
```

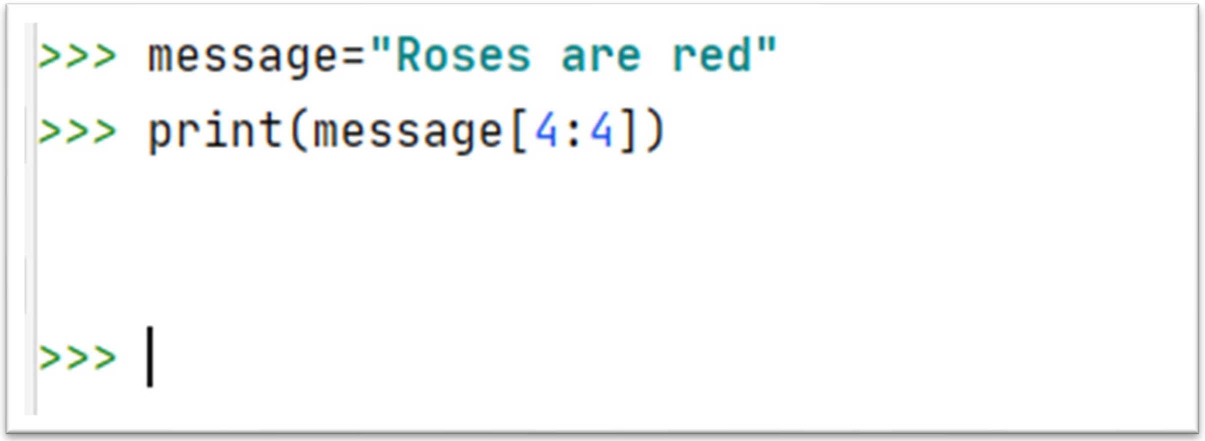
```
    print(message[6:9])
```

The above code will display: are

The 6 is included; the 9 is not.

String Slicing

- The operator [n:m] returns the part of the string starting from n index inclusive to m index exclusive. **n should always be less than m.**
- If the first index is the same as the last index the result is an empty string, an empty string (‘’) is a string with 0 characters.
- Example:



```
>>> message="Roses are red"
>>> print(message[4:4])


>>> |
```

String Slicing

- Negative numbers can be used as indexes to identify character positions relative to the end of the string.
- The Python interpreter adds the negative index to the length of the string to determine the position of the character.
- Index[-1] identifies the last character in a string,[-2] identifies the next to last character and so forth.

my_str = “THIS IS A TEST!”

T	H	I	S		I	S		A		T	E	S	T	!
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

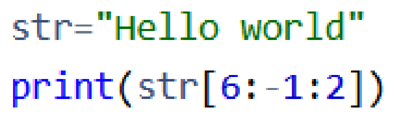
String Slicing

T	H	I	S		I	S		A		T	E	S	T	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
my_str = "THIS IS A TEST!"

my_str[0:4]   # "THIS"
my_str[:4]    # "THIS"
my_str[0:7]   # "THIS IS"
my_str[:7]    # "THIS IS"
my_str[10:14] # "TEST"
my_str[10:]   # "TEST!"
my_str[:-1]   # "THIS IS A TEST" – all but the last character;
              # chops off the final character
```

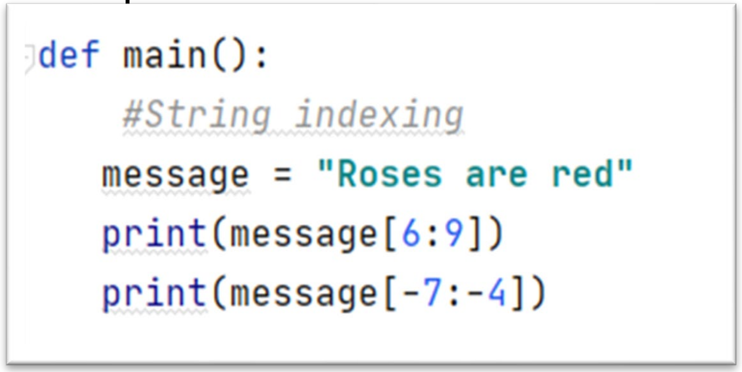
Practice: Output?



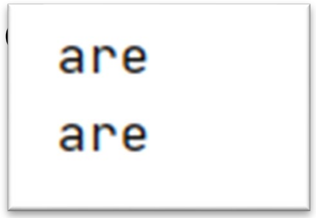
```
str="Hello world"
print(str[6:-1:2])
```

String Slicing

Example:



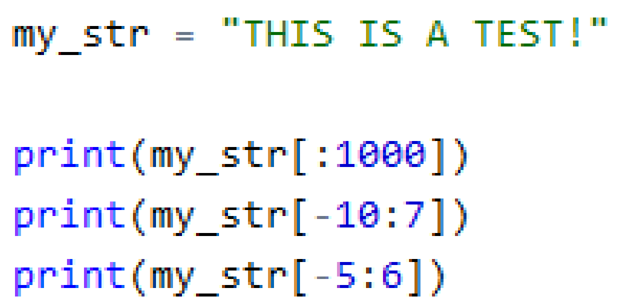
```
def main():
    #String indexing
    message = "Roses are red"
    print(message[6:9])
    print(message[-7:-4])
```



```
are
are
```

Note: if an index number is out of bounds, it is ignored, and the slice will use the start or the end index.

Practice: Output?



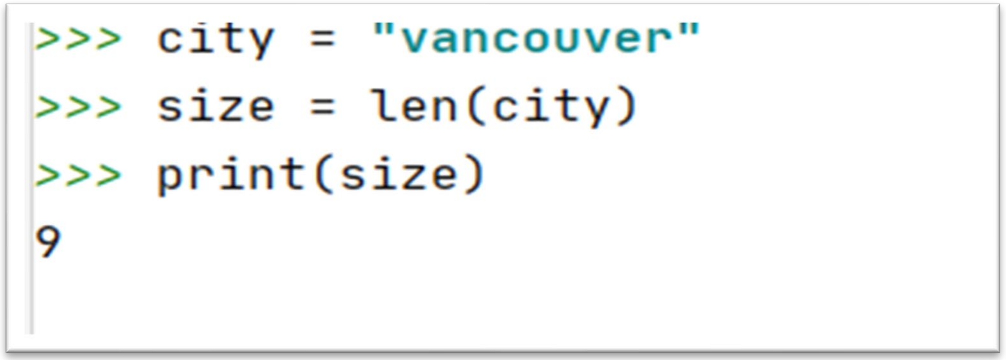
```
my_str = "THIS IS A TEST!"

print(my_str[:1000])
print(my_str[-10:7])
print(my_str[-5:6])
```

String Functions: len()

- Built-in function len() returns the length (number of characters) of a string.

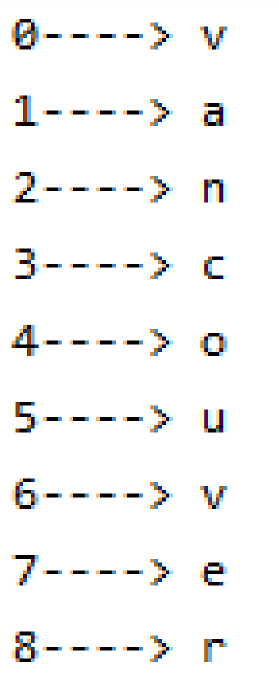
Example:



```
>>> city = "vancouver"
>>> size = len(city)
>>> print(size)
9
```

- Practice: To create a function that prints out each value and its associated index of string

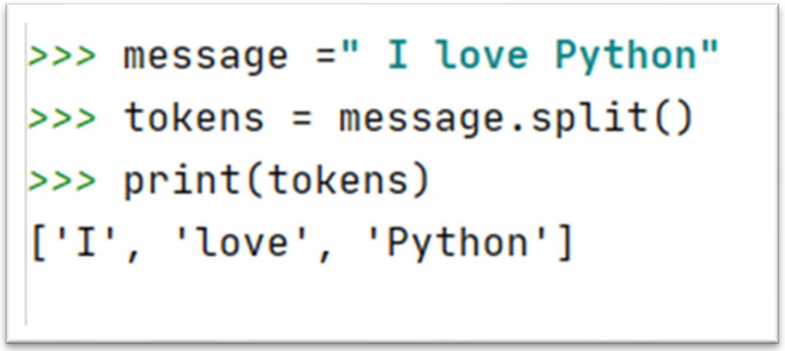
def print_values_with_index(string_value):



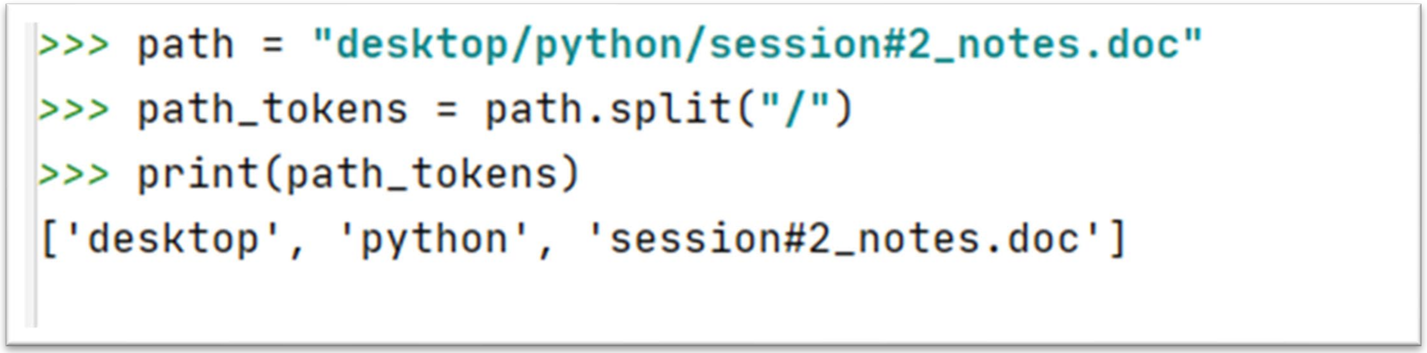
```
0----> v
1----> a
2----> n
3----> c
4----> o
5----> u
6----> v
7----> e
8----> r
```

String Methods: split()

- Method split splits the string into a list based on a separator.
- Examples:



```
>>> message =" I love Python"
>>> tokens = message.split()
>>> print(tokens)
['I', 'love', 'Python']
```



```
>>> path = "desktop/python/session#2_notes.doc"
>>> path_tokens = path.split("/")
>>> print(path_tokens)
['desktop', 'python', 'session#2_notes.doc']
```

String Methods: split()

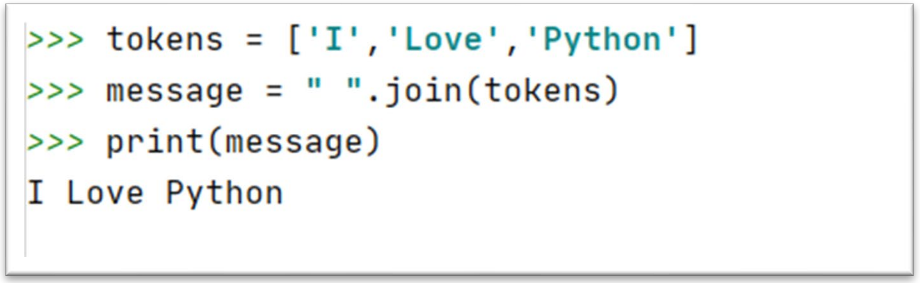
```
m = "I love Python"
tokens = m.split(" ")
print(tokens)
m = "I love Python"
tokens = m.split("o")
print(tokens)
```

Output:

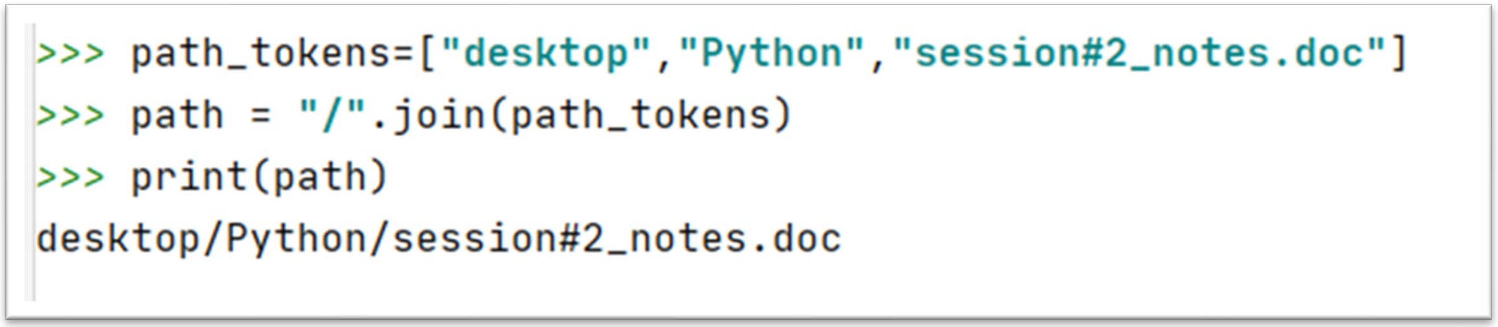
```
['I', 'love', 'Python']
['I l', 've Pyth', 'n']
```

String Methods: join()

- Method join is opposite of split, it joins a list of strings with a separator.
- Examples:



```
>>> tokens = ['I','Love','Python']
>>> message = " ".join(tokens)
>>> print(message)
I Love Python
```



```
>>> path_tokens=["desktop","Python","session#2_notes.doc"]
>>> path = "/".join(path_tokens)
>>> print(path)
desktop/Python/session#2_notes.doc
```

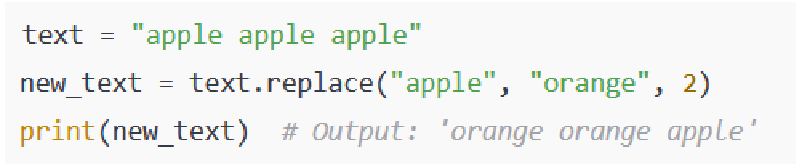
String Methods: join()

```
tokens = ["I", "love", "Python"]
m = "!!!".join(tokens)
print(m)
```

Output:

```
I!!!love!!!Python
```

More String Methods



```
text = "apple apple apple"
new_text = text.replace("apple", "orange", 2)
print(new_text)  # Output: 'orange orange apple'
```

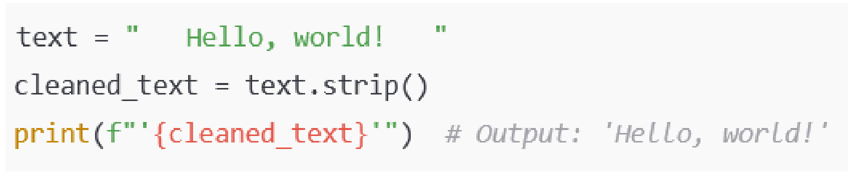
- ***replace(old, new)*** -- Returns a copy of the string with all occurrences of the substring **old** replaced by the string **new**. The old and new arguments may be string variables or string literals.
- ***replace(old, new, count)*** -- Same as above, except only replaces the first count occurrences of old.
- ***find(x)*** -- Returns the position of the first occurrence of item x in the string, otherwise returns -1, if x was not in the string. x may be a string variable or string literal.
- ***find(x, start)*** -- Same as find(x), but begins the search at position start
- ***find(x, start, end)*** -- Same as find(x, start), but stops the search at position end
- ***rfind(x)*** -- Same as find(x) but searches the string in reverse, returning the last occurrence in the string.
- ***count(x)*** -- Returns the number of times x occurs in the string.

String Methods: boolean return

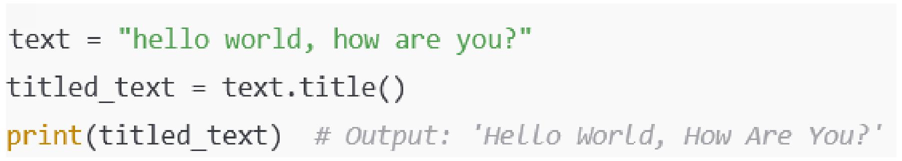
- ***isalnum()*** -- Returns True if all characters in the string are lowercase or uppercase letters, or the numbers 0-9.
- ***isdigit()*** -- Returns True if all characters are the numbers 0-9.
- ***islower()*** -- Returns True if all characters are lowercase letters.
- ***isupper()*** -- Returns True if all cased characters are uppercase letters.
- ***isspace()*** -- Returns True if all characters are whitespace.
- ***startswith(x)*** -- Returns True if the string starts with x.
- ***endswith(x)*** -- Returns True if the string ends with x.

String Methods: string return

- ***capitalize()*** -- Returns a copy of the string with the first character capitalized and the rest lowercased.
- ***lower()*** -- Returns a copy of the string with all characters lowercased.
- ***upper()*** -- Returns a copy of the string with all characters uppercased.
- ***strip()*** -- Returns a copy of the string with leading and trailing whitespace removed.
- ***title()*** -- Returns a copy of the string as a title, with first letters of words capitalized.



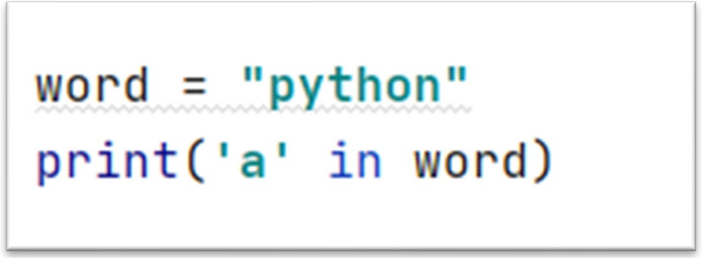
```
text = "   Hello, world!   "
cleaned_text = text.strip()
print(f"'{cleaned_text}'")  # Output: 'Hello, world!'
```



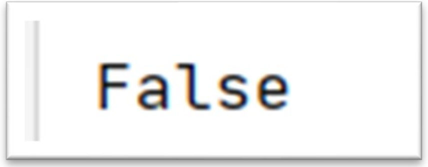
```
text = "hello world, how are you?"
titled_text = text.title()
print(titled_text)  # Output: 'Hello World, How Are You?'
```

In Operator

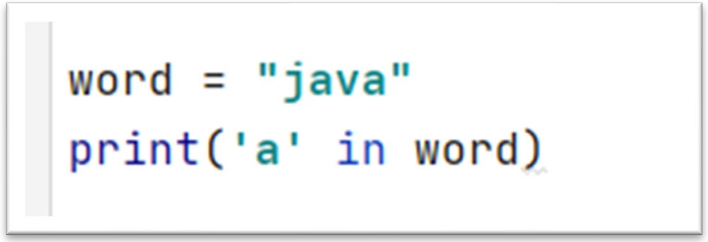
- The in operator is used to check if an item exists in a sequence.
- The in operator can be used to check if a substring exists in another string.
- Examples:



```
word = "python"
print('a' in word)
```

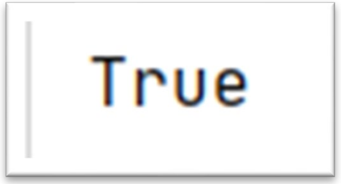


```
False
```



```
word = "java"
print('a' in word)
```

Output:



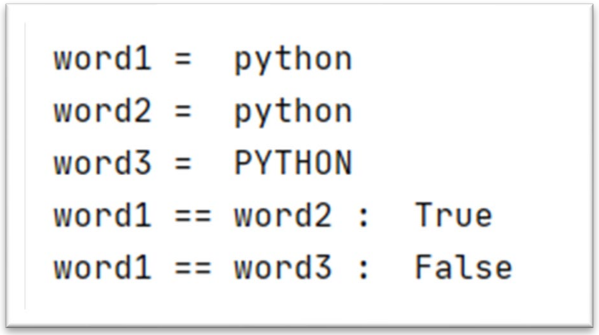
```
True
```

Comparing Strings

Example	Expression result	Why?
'Hello' == 'Hello'	True	The strings are exactly identical values
'Hello' == 'Hello!'	False	The left-hand string does not end with '!'
'Yankee Sierra' > 'Amy Wise'	True	The first character of the left side 'Y' is "greater than" (in ASCII value) the first character of the right side 'A'
'Yankee Sierra' > 'Yankee Zulu'	False	The characters of both sides match until the second word. The first character of the second word on the left 'S' is not "greater than" (in ASCII value) the first character on the right side 'Z'
'seph' in 'Joseph'	True	The substring 'seph' can be found starting at the 3rd position of 'Joseph'
'jo' in 'Joseph'	False	'jo' (with a lowercase 'j') is not in 'Joseph' (with an uppercase 'J')

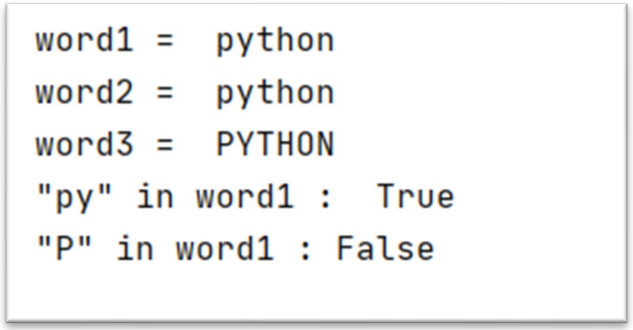
String Comparison

- Operator == is used to compare the equality of two String values
- Example:



```
word1 =  python
word2 =  python
word3 =  PYTHON
word1 == word2 :  True
word1 == word3 :  False
```

- The <u>in</u> operator is used to check if one string is a substring of another.
- Example:

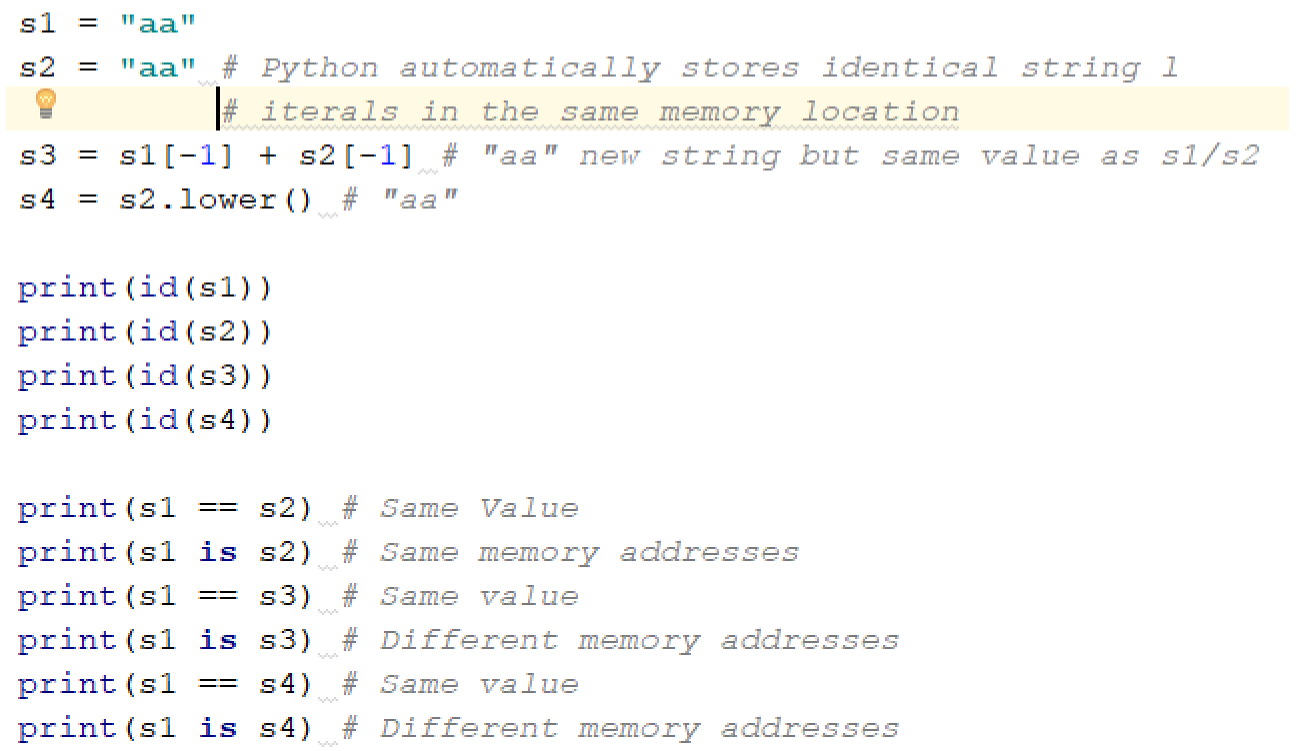


```
word1 =  python
word2 =  python
word3 =  PYTHON
"py" in word1 :  True
"P" in word1 : False
```

is

- The is operator checks memory addresses of the String objects, not the letter contents of the string.
- Example:

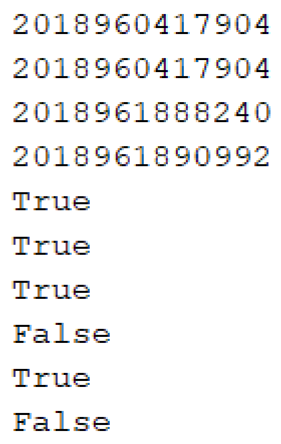
Output:



```
s1 = "aa"
s2 = "aa" # Python automatically stores identical string l
          # iterals in the same memory location
s3 = s1[-1] + s2[-1] # "aa" new string but same value as s1/s2
s4 = s2.lower() # "aa"

print(id(s1))
print(id(s2))
print(id(s3))
print(id(s4))

print(s1 == s2) # Same Value
print(s1 is s2) # Same memory addresses
print(s1 == s3) # Same value
print(s1 is s3) # Different memory addresses
print(s1 == s4) # Same value
print(s1 is s4) # Different memory addresses
```



```
2018960417904
2018960417904
2018961888240
2018961890992
True
True
True
False
True
False
```

Lab 4 and Next Week's Homework

Lab 4 has two parts:

- Part A – To be done in class. Show me you running code tonight and include the data_format.py and main.py files with your Part B submission.
- Parts B and C – To be done at home (or in class if you finish Part A early). Due in the Lab 4 Dropbox Wednesday at midnight

I will start you off on Part A and you can complete and demo it before you leave tonight.