

# Python for Text Analysis

## 2018-2019

---

Lecture 2: Strings, Booleans and If-statements  
01-11-2018

# Today's class

## ❖ **First half:** Theory

- Questions so far?
- Some notes about Notebooks
- Chapter 3: Strings
- Chapter 4: Boolean Expressions and Conditions

## ❖ **Second half:** Working on assignment

- Room MF-A307
- Room MF-A115 (optional)

**Slides** can be found on **Github > Slides** and on **Canvas > Files**.

# SOCRATIVE

**Log in at:**

[b.socrative.com/student](https://b.socrative.com/student)

**Room:**

PYTHON1819VU

# Some notes about Notebooks

- ❖ Behind every notebook runs a **kernel** (can be seen as the whole document)
  - When you run a code cell, that code is executed within the kernel
  - Any output is returned back to the cell to be displayed
- ❖ The kernel's state persists **over time** and **between cells**
  - If you **declare variables** in one cell, they will be **available** in another
  - Similarly, these variables will **remain in memory** even if you **renamed variables** and execute the cell again (this happens a lot...!)
- ❖ To **test your code** (or possibly fixing problems that seem unrelated to the code in your cell), it may be useful to reset things:
  - **Restart:** restarts the kernel, thus clearing all the variables etc. that were defined.
  - **Restart & Clear Output:** same as above but will also wipe the output displayed.
  - **Restart & Run All:** same as above but will also run all your cells in order from first to last.



## CHAPTER 3:

### Strings

---

# Strings

- ❖ Strings are created with **quotes** (‘ or “)
- ❖ Strings are **sequences** of characters
  - Each character has a **positive index** and a **negative index**
- ❖ Strings are **immutable**
  - You are **not** allowed to **modify** the individual characters in the string

```
my_string = "Hello"  
my_string = 'Hello'
```

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

# String indices

- ❖ We can **access characters** in a string using their **indices**
  - **First character:** index = 0
  - **Last character:** index = -1

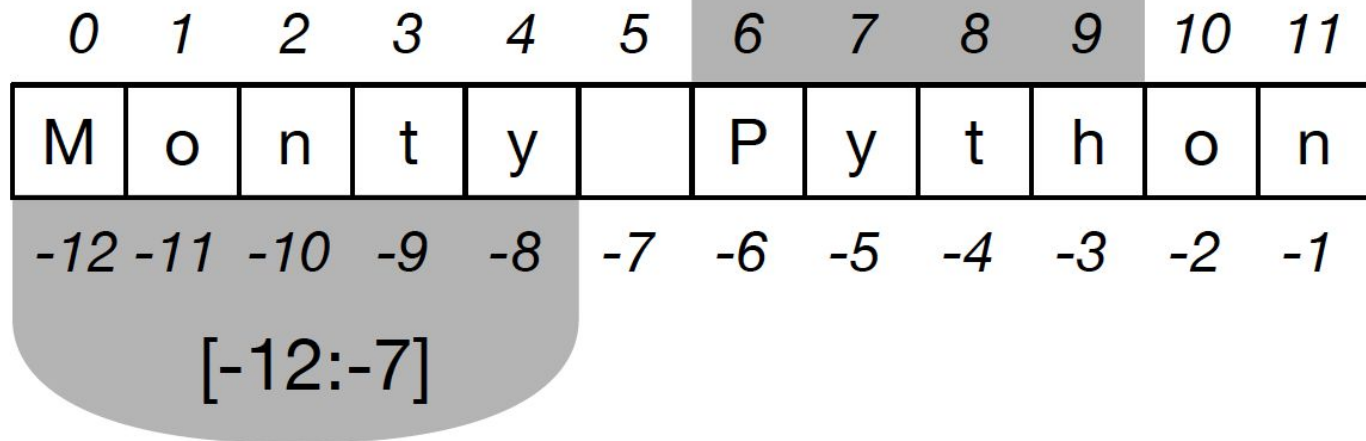
```
my_string[0] == "H"  
my_string[1] == "e"  
my_string[-4] == "e"
```

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

# String slicing

- ❖ We can also extract a **range** from a string

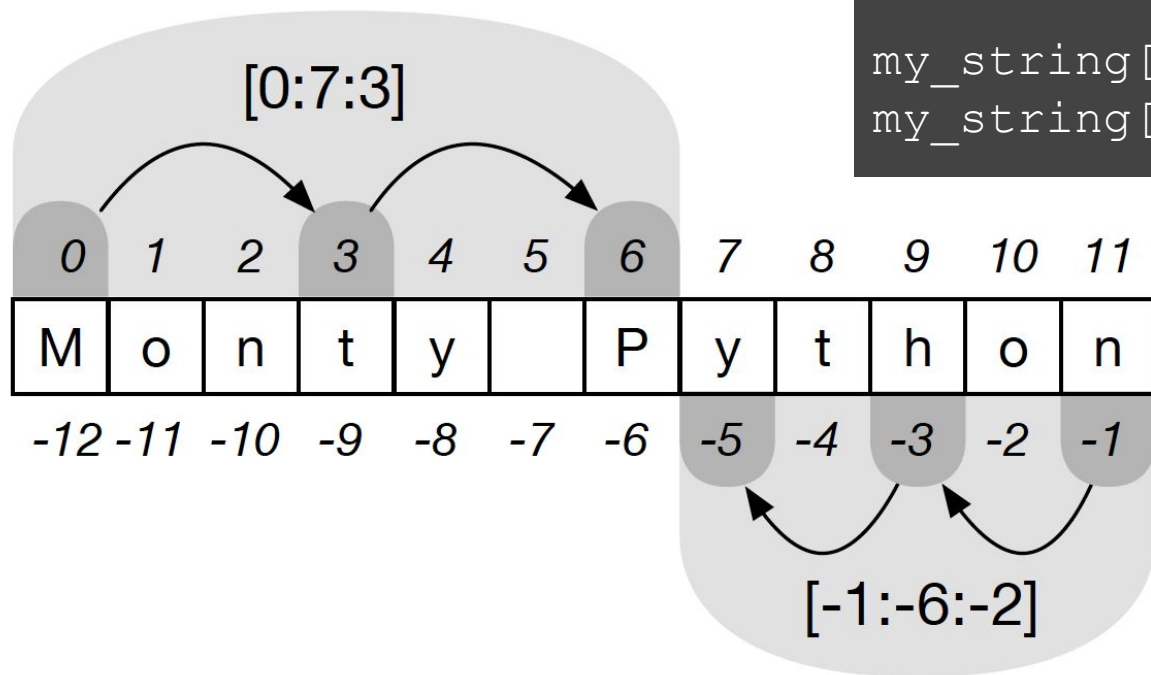
```
my_string[6:10] == "Pyth"  
my_string[-12:-7] == "Monty"
```





# String slicing

- ❖ We can also extract a **range** from a string using a specific **step size**



```
my_string[0:7:3]      == "MtP"
my_string[-1:-6:-2]   == "nhy"
```

# String are immutable

- ❖ You are **not** allowed to **modify** the individual characters in the string
- ❖ Instead, you should create a **new string** based on the **old string**
  - For example, by using `str.replace()` or by concatenation

```
my_string = "cat"  
my_string[0] = "f"
```

```
-----  
TypeError                                 Traceback (most recent call last)  
<ipython-input-4-5b2a04c5d3c4> in <module>()
```

```
    1 my_string = "cat"  
----> 2 my_string[0] = "f"
```

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

```
my_string = "cat"  
my_string = "f" + my_string[1:]  
print(my_string)
```

```
fat
```

# SOCRATIVE

**Log in at:**

[b.socrative.com/student](https://b.socrative.com/student)

**Room:**

PYTHON1819VU

# String concatenation and formatting

## ❖ Sometimes we want to **combine** strings

- for printing
- for creating a new string

## ❖ Several ways to achieve this:

- using the **print() function with commas**
- **concatenating** strings using:
  - the + sign
  - the \* sign
- **interpolating** strings using **string formatting**:
  - f-strings (recommended for Python 3.6+)
  - %-formatting
  - .format()

# Printing multiple objects with commas

```
first_name = "Monty"  
second_name = "Python"  
print(first_name, second_name)
```

Monty Python

```
number = 3  
product = "oranges"  
print(number, product)
```

3 oranges

`print()` function

- + prints all objects to the screen separated by space (default)
- + does not create a new string

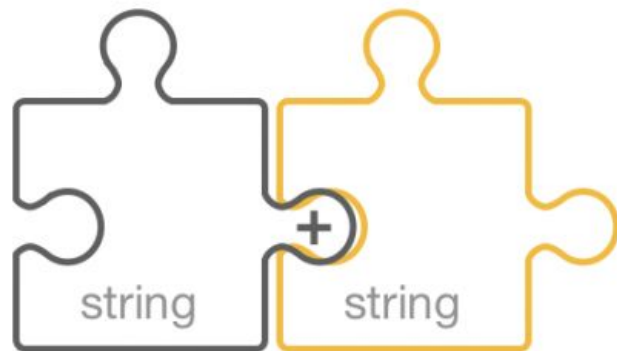
# Concatenating strings using the + sign

```
first_name = "Monty"  
last_name = "Python"  
full_name = first_name + last_name  
print(full_name)
```

MontyPython

```
name = "Monty"  
name += "Python"  
print(name)
```

MontyPython



- + makes a new string
- + does not use a delimiter

# Concatenating strings using the + sign

```
x = 20
y = "18"
print(str(x) + y)
```

2018

```
x = 20
y = "18"
print(x + int(y))
```

38

```
x = 20
y = "18"
print(x + y)
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-29-27f7d79bbd92> in <module>()
      1 x = 20
      2 y = "18"
----> 3 print(x + y)
```

**TypeError:** unsupported operand type(s) for +: 'int' and 'str'

+ is interpreted based on type

## Repeating strings using the \* sign

```
x = 3  
y = 'very '  
z = 'hot'  
print(x * y + z)
```

very very very hot

```
x = 3  
y = 5  
print(x*y)
```

\* is interpreted based on type



# Interpolating strings using f-strings (string formatting)

```
n = 3  
print(f"{n} oranges, please")
```

3 oranges, please

```
n = 3  
price = 0.25  
print(f"That will be €{n*price}, then")
```

That will be €0.75, then

## f-strings

- can execute code in-place
- convert all objects to strings

# SOCRATIVE

**Log in at:**

[b.socrative.com/student](https://b.socrative.com/student)

**Room:**

PYTHON1819VU

# String functions and methods

<code>len(s)</code>	Returns the length (number of characters) of string <code>s</code>
<code>s.find(x)</code>	Find the first position of substring <code>x</code> in string <code>s</code>
<code>s.count(x)</code>	Count the number of times substring <code>x</code> is in string <code>s</code>
<code>s.upper()</code> <code>s.lower()</code>	Returns a string that is all uppercase or lowercase
<code>s.replace(x, y)</code>	Returns a new string where the substring <code>x</code> has been replaced by the substring <code>y</code>
<code>s.startswith(x)</code> <code>s.endswith(x)</code>	Returns <code>True</code> if string <code>s</code> starts/ends with substring <code>x</code> , otherwise returns <code>False</code>

# String methods

- ❖ Some methods take **input arguments**, others do not
- ❖ More about arguments in block 2
- ❖ For now, remember that:
  - you can **call `help()`** to find out about input arguments
  - Python will throw a **`TypeError`** if you don't provide an **obligatory argument**

```
my_string = "hello"  
my_string.count()
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-7-afa5043439f0> in <module>()  
      1 my_string = "hello"  
----> 2 my_string.count()
```

```
TypeError: count() takes at least 1 argument (0 given)
```

```
help(str.count)
```

Help on method\_descriptor:

```
count(...)
  S.count(sub[, start[, end]]) -> int
```

Return the number of non-overlapping occurrences of substring **sub** in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

*what to count?*

# String methods

- ❖ String methods only **return** the result; they do **not change** the string itself
  - Remember: strings are **immutable**

```
my_string = "cat"  
my_string.replace("c", "f")  
print(my_string)
```

cat

```
my_string = "cat"  
my_string = my_string.replace("c", "f")  
print(my_string)
```

fat





# SOCRATIVE

**Log in at:**

[b.socrative.com/student](https://b.socrative.com/student)

**Room:**

PYTHON1819VU

$\&\&$	!alive	alive
!dead		
dead		

## CHAPTER 4:

# Boolean Expressions and Conditions

---

# Boolean Expressions and Conditions

- ❖ There are two Boolean values: **True** and **False** (type = bool)
- ❖ **Boolean expressions** result in one of these

RESULTING IN **TRUE**

```
3 == 3  
3 < 5  
"H" in "Hello"
```

RESULTING IN **FALSE**

```
3 == 5  
3 > 5  
"P" in "Hello"
```

- ❖ Boolean expressions can be used as **conditions** in **if-elif-else constructs**



# Comparison operators

- ❖ **Comparison operators** compare the **values on either sides** of them and decide the **relation** among them

operator	meaning	True	False
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

# Membership operators

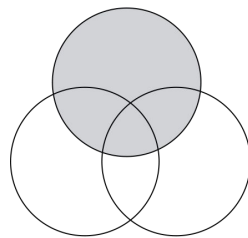
- ❖ **Membership operators** test for **membership** in a **sequence**, such as strings, lists, or tuples

operator	meaning	True	False
<code>in</code>	left object is a member of right object	<code>"fun" in "function"</code>	<code>"I" in "team"</code>
<code>not in</code>	left object is NOT a member of right object	<code>"I" in "team"</code>	<code>"fun" in "function"</code>

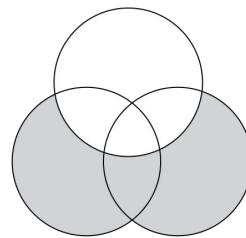
# AND, OR & NOT

operator	meaning	True	False
bool1 <b>and</b> bool2	True if both <b>bool1</b> and <b>bool2</b> are True, otherwise False	5 == 5 and 3 < 5	5 == 5 and 3 > 5
bool1 <b>or</b> bool2	True when at least one of the boolean expressions is True, otherwise False	5 == 5 and 3 > 5	5 != 5 and 3 > 5
<b>not</b> bool1	True if bool1 is False, otherwise False	not 5 != 5	not 5 == 5

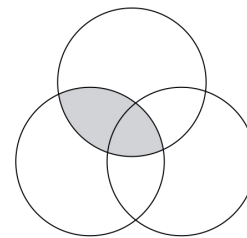
# AND, OR & NOT



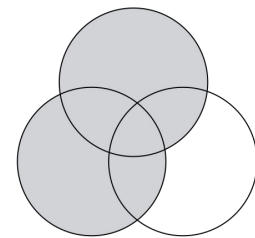
A



Not A

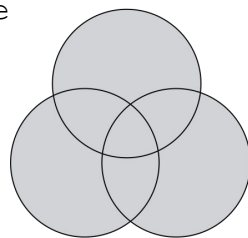


A And B

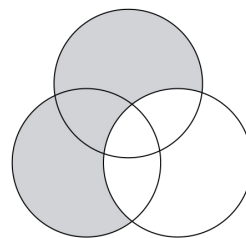


A Or B

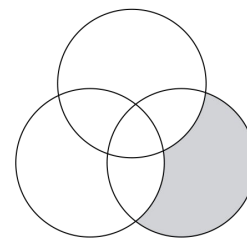
A = "Johnny Depp" in movie  
 B = "Helena Bonham-Carter" in movie  
 C = "Alan Rickman" in movie



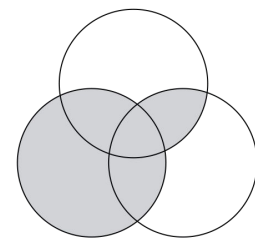
A Or B Or C



(A Or B) and Not C

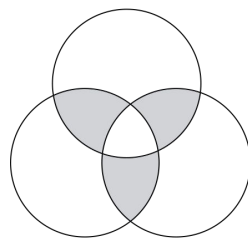


C And Not A and Not B

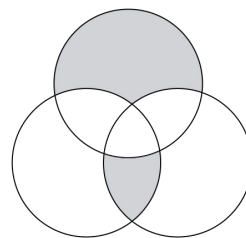


B Or (C And A)

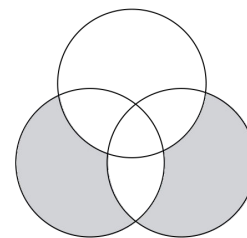
Can you understand which (hypothetical) movies  
 each boolean expression would express?



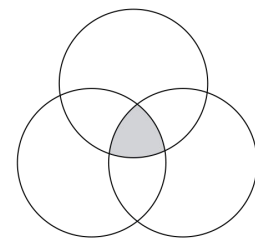
(A And B) Or  
 (B And C) Or (A And C)



((B And C) and Not A) Or  
 (A and Not B and Not C)

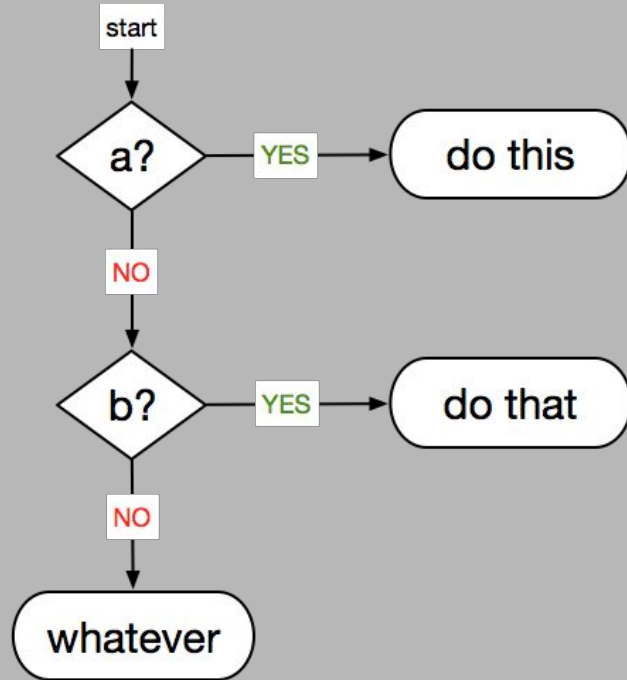


(B And Not C) Or  
 (C and Not B)



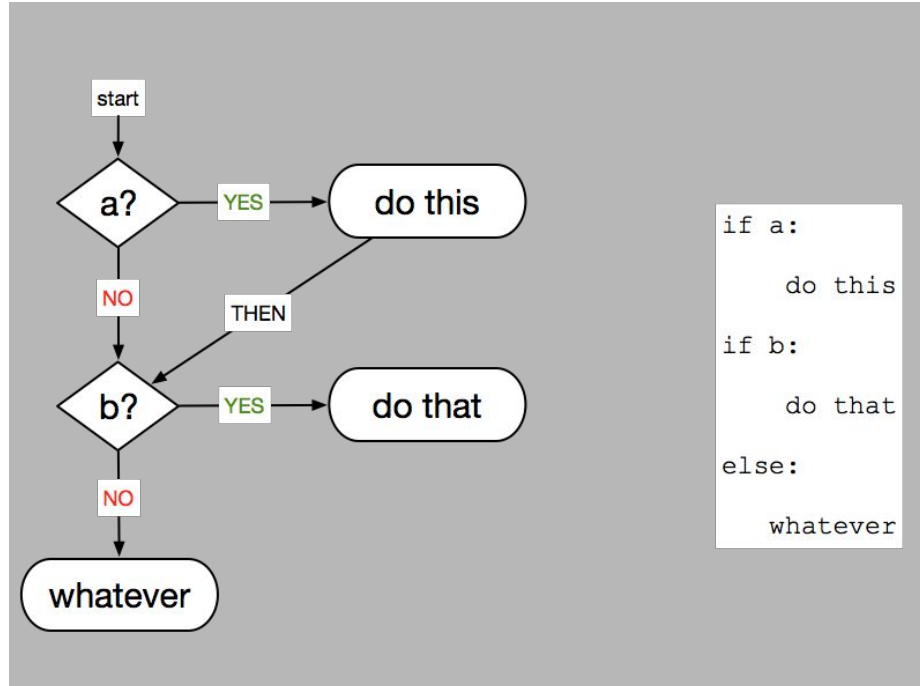
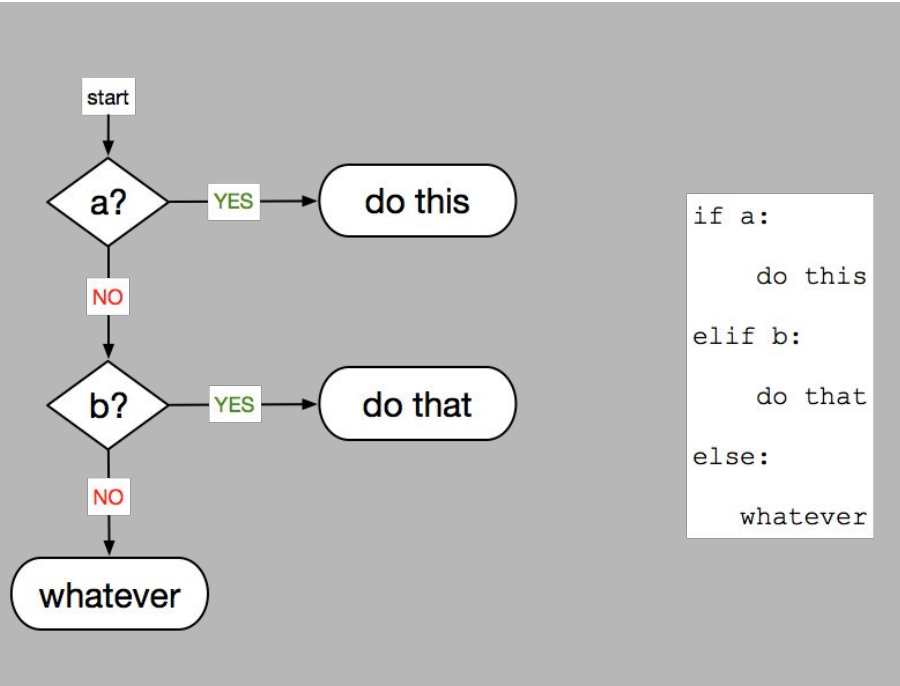
A And B And C

# If - elif - else

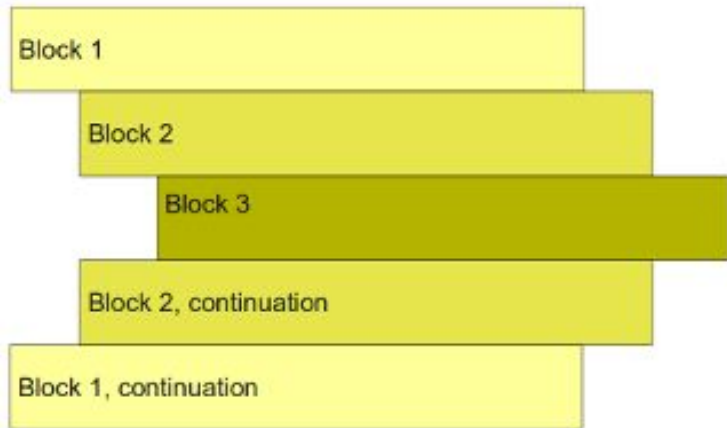


```
if a:  
    do this  
elif b:  
    do that  
else:  
    whatever
```

# Difference between if-elif and if-if



# Indentation & Nesting



**DON'T DO:**

```
if "h" in "hello":  
    print("bye")
```

**INSTEAD, DO:**

```
if "h" in "hello":  
    print("bye")
```

# SOCRATIVE

**Log in at:**

[b.socrative.com/student](https://b.socrative.com/student)

**Room:**

PYTHON1819VU



# Next week

- ❖ **Tomorrow before 23:59:** submit Assignment 1
- ❖ **Monday:** feedback session (no preparation needed)
- ❖ **Thursday:** start with Block 2