Introduction to Computation for the Humanities and Social Sciences

CS 3

Chris Tanner

Lecture 6

```
if i_were_a_boy:
```

— Beyoncé

REFRESHER:

Try/Except statements are used when we know that our data could potentially have undesired/ unpredictable behavior (e.g., the data is coming from the user, who has freedom to type unintended input).

Instead of our program crashing with an error, we **Try** to do our regular code that we wish, but we also aim to display **Exceptions** to our expectations, in which case we can execute other code such as informing the user how we wish for them to correctly use our program.

REFRESHER:

We have 2 main types of numeric data types:

Ints (for whole numbers)

Floats (for when we care about having decimal precision like 4.9)

We should use whichever one is most appropriate for the given data that we care to store. e.g., weight should be a **float**. An age variable could be a **Int**, because we rarely talk about being 19.6 years old.

When you initialize a variable, you don't need to specify the *type*, Python can infer it:
e.g., **x** = 19 vs **x** = 19.4

Only when you start to do operations do you possibly need to be explicit to ensure the data is of the *type* you want.

Lecture 6

- Control Flow: if-statements
- Comparison Operators
- HW4: Rock, Paper, Scissors
- Project 1

Control Flow — **if-statements**

- Often times, it's useful to write code which will only be executed under certain conditions — i.e., something being true or not.

- Ex: Designing a program that calculates statistics for senators, but the user gets to pick if they are interested in **democratic** or **republican** senators.

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #1

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #2

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #2

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #2

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-statements**

Scenario #2

```
lines of code …
if <something is true>:
    execute these lines of code

more lines of code
```

Control Flow — **if-sta**

```
lines of
if <some
    exec

    more
```

- These lines get executed **regardless** of the if-statement being true or not.
- Nothing inherently wrong w/ that, but sometimes we wish to execute code ONLY when the if-statement is **false**
- behold: the **`else`** statement

Control Flow — **if-statements**

Scenario #3

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #3

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #3

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #3

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #3

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Scenario #4

```
lines of code …
if <something is true>:
    execute these lines of code
else:
    execute these other lines

more lines of code
```

Control Flow — **if-statements**

- That's cool, but sometimes we have more than just 2 scenarios (i.e., true or false) that we wish to handle.
- e.g., Handling political parties that could be Democrat, Republican, Independent, or an invalid entry
- behold: the **`elif`** statement
- short for "else if"

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Control Flow — **if-statements**

Scenario #5

```
lines of code …
if <something is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
elif <something else is true>:
    execute these lines of code
else:
    execute these other lines
more lines of code
```

Scenario #5

For statements with more than one **elif**, structure code so that the **else** block handles the unexpected or invalid cases

```
        execute these lines of code
    else:
        execute these other lines
    more lines of code
```

Lecture 6

- Control Flow: if-statements
- Comparison Operators
- HW4: Rock, Paper, Scissors
- Project 1

Comparison Operators

- To check if something is **`True`** or **`False`**, we need to compare some values that we care about.

- Comparison operators *compare* the right-hand side and the left-hand side

- Every comparison results in a **`True`** or **`False`** answer

Comparison Operators

Equality

To check if two items are equal to each other, use **==**

Syntax:

```
a == b
```

Comparison Operators

Equality

```
political_party = input("Which political party are you interested in? ")
if political_party == "Democrat":
    execute these lines of code
elif political_party == "Republican":
    execute these lines of code
elif political_party == "Independent":
    execute these lines of code
else:
    print("ERROR: Please enter Democrat, Republican, \
        or Independent")
    exit(1)
more lines of code
```

Not Equals

To check if two items are **not** equal to each other, use **!=**

Syntax:

```
a != b
```

Examples:

```
if political_party != “Democrat”:
```

or

```
if 5 != 6:
```

Comparison Operators

equals:	`5 == 5`	**`True`**	`“hello” == “hello”`	**`True`**
not equals:	`5 != 6`	**`True`**	`“cat” != “dog”`	**`True`**
greater than:	`6 > 5`	**`True`**	`“help” < “hello”`	**`False`**
greater than or equal to:	`5 >= 5`	**`True`**	`“hello” >= “hello”`	**`True`**
less than:	`4 < 5`	**`True`**	`“hello” < “help”`	**`True`**
less than or equal to:	`5 <= 5`	**`True`**	`“hello” <= “hello”`	**`True`**

Comparison Operators

- Strings compare **alphabetically**
- To check if a **substring** is contained within a larger string, use **in**
- For any string comparisons, make sure they in the same case using **str.lower()**

eq	o" == "hello"	True
no	= "dog"	True
gre	< "hello"	False
gre	" >= "hello"	True
les	" < "help"	True
les	" <= "hello"	True

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print(“they are equal”)
else:
    print(“they are not equal”)
```

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print("they are equal")
else:
    print("they are not equal")
```

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print("they are equal")
else:
    print("they are not equal")
```

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print(“they are equal”)
else:
    print(“they are not equal”)
```

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print("they are equal")
else:
    print("they are not equal")
```

Comparison with Floats

- Remember float operations are only approximations
- If you want to check if the result of an expression equals a float value, check instead if their difference is small

```
if (0.1 + 0.2) == 0.3:
    print("they are equal")
else:
    print("they are not equal")
```

Comparison with Floats

- It's more common to test if a Float is greater than or less than a value, which isn't problematic.

- You rarely need to compare Floats for equality; however, if you wish to, use:

```
import math

a = 0.1 + 0.2
print(math.isclose(a,0.3))
```

Comparison with Floats

- It's more common to test if a Float is greater than or less than a value, which isn't problematic.

- You rarely need to compare Floats for equality; however, if you wish to, use:

```
import math

a = 0.1 + 0.2
print(math.isclose(a,0.3))
```

Substrings

```
if “og” in “dog”:
    print(“it contains the substring!”)
else:
    print(“it does not contain it!”)
```

Comparison Operators

Substrings

```
if “og” in “dog”:
    print(“it contains the substring!”)
else:
    print(“it does not contain it!”)
```

Comparison with Strings

- Strings compare **alphabetically**
- To check if a substring is contained within a larger string, use **in**
- For any string comparisons, make sure they in the same case using **str.lower()**

```
"i" in "team" -> False
"fun" in "function" -> True

a = "Fun"
b = "I'm having fun with functions"
print(a in b)                          -> False
print(a.lower() in b.lower())          -> True
```

Comparison Operators

Not, And, And/or, Or

- **not**, **and**, **or** are operators
- **and**: evaluates to True if the left-hand side is True and the right-hand side is True
- **or**: evaluates to True if either side is True
- **not**: inverts a True to False or vice versa
- For comparing more than two values, need to use multiple ands/ors

```
True and True -> True
True and False -> False

True or False -> True
False or False -> False

True and not False -> True
“fun” not in “function” -> False

a and b and c
a or b or c
(a or b) and (c or d)
```

String Special Characters

- **Some characters in strings have special meanings**
- **You can tell them apart because they start with a backslash**

```
print("He said, \"Hello.\"")
He said, "Hello."

print('He didn\'t go to the store')
He didn't go to the store

print("This text is two lines.\nThis is line 2.")
This text is two lines.
This is line 2.
```

String Special Characters

Character	Meaning
\t	tab
\n	new line (Mac)
\r\n	new line (Windows)
\\	\
\'	'
\"	"

Methods vs Functions

- **Methods** are functions that operate on a specific variable or value
- Syntax is variable.method_name()
- The method performs computations specific to the variable

```
name = “Chris”
print(name.lower())
print(name.upper())

print(name.replace(‘r’, ‘x’))
```