



CoGrammar

Lecture 2: Data Types and Conditional Statements

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(FBV: Mutual Respect.)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Open Classes.
You can submit these questions here: [Open Class Questions](#)

Data Science Lecture Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Lecture Objectives

- **Recap on data types**
- **Understanding basic functions
conditional statements.**

Let's recap!

- ★ **Python has 4 data types:**
 - **Integers**
 - **Floats**
 - **Strings**
 - **Booleans**
- ★ **We can store these as variables to use in other operations in our code.**
- ★ **We're going to go over them again briefly to ensure we're all up to speed what each of those are along with some key methods we can use with each.**

The String Data Type

- ★ Strings in Python are detected by quotation marks (""") or inverted commas (")
- ★ Example :

```
quotation_str = "The quick brown fox jumps over the lazy dog"  
inverted_comma_str = 'Strings are rather useful, what do you think?'
```

Concatenation of Strings

- ★ Strings can be added to one another. This is referred to as concatenation.
- ★ Example :

```
name = "Pieter"  
surname = "Parker"  
  
full_name = name + surname  
full_name = name + " " + surname
```

What are Methods?

- ★ **Methods are ways to express and action in programming.**
 - **Within the brackets of the method are its arguments.**
 - **Arguments are extra information given to the method.**
 - **Data types like strings have built-in methods we can use to either make changes to the string or find out more about it.**

len()

- ★ The len() function will simply output the length value of a string in the form of an int value.
(This will also work with lists and dictionaries, but we'll cover those later on in this bootcamp)

- ★ Example:

```
message = "batman"  
message_len = len(message)  
print(message_len)  
  
# Result >> 6
```

upper()

- ★ The upper() method will take a string and convert all the characters to uppercase.
- ★ **NOTE:** We have to use dot notation here because it's a part of the string data type.

★ Example:

```
message = "PyTh0n Is FuN"  
new_message = message.upper()  
print(new_message)  
  
# Result >> "PYTHON IS FUN"
```

lower()

- ★ The lower() method will take a string and convert all the characters to lowercase.
- ★ Similar to “upper()”, we also need to use dot notation here.
- ★ Example:

```
message = "PyThOn Is FuN"  
new_message = message.lower()  
print(new_message)  
  
# Result >> "python is fun"
```

capitalize()

- ★ The `capitalize()` method will take a string and convert the first letter to uppercase and the rest of the characters to lowercase, should there be any other uppercase characters.
- ★ Example:

```
message = "PyThOn Is FuN"  
new_message = message.capitalize()  
print(new_message)  
  
# Result >> "Python is fun"
```

strip()

- ★ The strip() method will remove a symbol from a string.
- ★ Keep in mind that strip() will only remove from the ends of a string.
- ★ Example:

```
message = "****They've*taken*the*hobbits*to*Eisenguard!****"  
message_strip = message.strip("*")  
print(message_strip)  
  
# Result >> "They've*taken*the*hobbits*to*Eisenguard"
```

split()

- ★ The `split()` method, will split a string by a symbol. However, once the split occurs the string will then be placed in what's called a list, which can be indexed.
- ★ Example:

```
message = "The-king-of-iron-fist"  
message_split = message.split("-")  
print(message_split)  
  
# Result >> ["The", "king", "of", "iron", "fist"]
```

join()

- ★ The `join()` method will take a list containing only string values, and concatenate them to form one string. (We will cover lists in more detail later on in the bootcamp)
- ★ Example:

```
list_example = ["The", "king", "of", "iron", "fist"]  
list_join = " ".join(list_example)  
print(list_join)  
  
# Result >> "The king of iron fist"
```

replace()

- ★ The `replace()` method will replace any specified character in a string with a new one. Keep in mind that `replace()` requires two arguments to function. First to identify what to replace, and second to identify what to replace it with.
- ★ Example:

```
message = "Hey!you!over!there!"  
message_replace = message.replace("!", " ")  
print(message_replace)  
  
# Result >> "Hey you over there"
```


Indexing

- ★ Strings are basically a list of characters. An example would be “Hello”, which consists of the characters H+e+l+l+o.

Hello

0 1 2 3 4

-5 -4 -3 -2 -1

String Slicing

- ★ String slicing is a way of extracting multiple characters from a string based on their index position.
- ★ Important to remember that this is done character by character, not word by word.
- ★ Example:

```
string = "Hello"
string_idx = string[3]
print(string_idx)

# Result >> "l"

string_slice = string[0:3]
print(string_slice)

# Result >> "Hel"
```

Escape Characters

- ★ Python uses the backslash (\) as an escape character.
- ★ The backslash is used as a marker to inform the compiler that the next character has a special use/meaning.
- ★ The backslash combined with specific other characters is known as an escape character.

Escape Characters

- ★ **Some useful escape characters:**
 - `\n` - New line
 - `\t` - Tab Space
- ★ **The escape character can also be used for quoting in a string.**
- ★ **By placing a backslash in front of a quotation mark, you can tell the compiler to avoid terminating the string.**

Let's Breathe

Let's take a small break before moving on to the remaining data types.

Recap on numbers in Python

- ★ Here we will recap on the two basic types of numbers used in Python:
 - Integers : whole numbers that are either positive or negative :
 - E.g. -32, 0, 600, 138227, etc.
 - Floats : decimal numbers that are also either positive or negative:
 - E.g. 6.2, -27.157, 33.3333, etc.

Declaring Numeric Variables

- ★ Python is able to determine what data type a variable is based on the data's characteristics:
- ★ `num_one = 7` → no decimal point, no quotation marks, meaning it has to be an integer.
- ★ `avg_grade = 8.3` → decimal point, no quotation marks, meaning it has to be float.

Arithmetic Operations

Similarly, with real world mathematics, we are able to apply maths to our numeric variables.


However, note that Python has a different way of interpreting the operation symbol, meaning that multiplication in Python is not written as 'x'. The same applies for division and exponents.

Arithmetic Operations

```
addition = 6 + 2  
# Result >> 8  
  
subtraction = 6 - 2  
# Result >> 4  
  
multiplication = 9 * 3  
# Result >> 27  
  
division = 12 / 3  
# Result >> 4  
  
modulus = 9 % 3  
# Result >> 0  
  
exponential = 6 ** 2  
# Result 36
```




What is the index of “e” in the string “hello”?

- A. 0
 - B. 2
 - C. -1
 - D. 1
- 

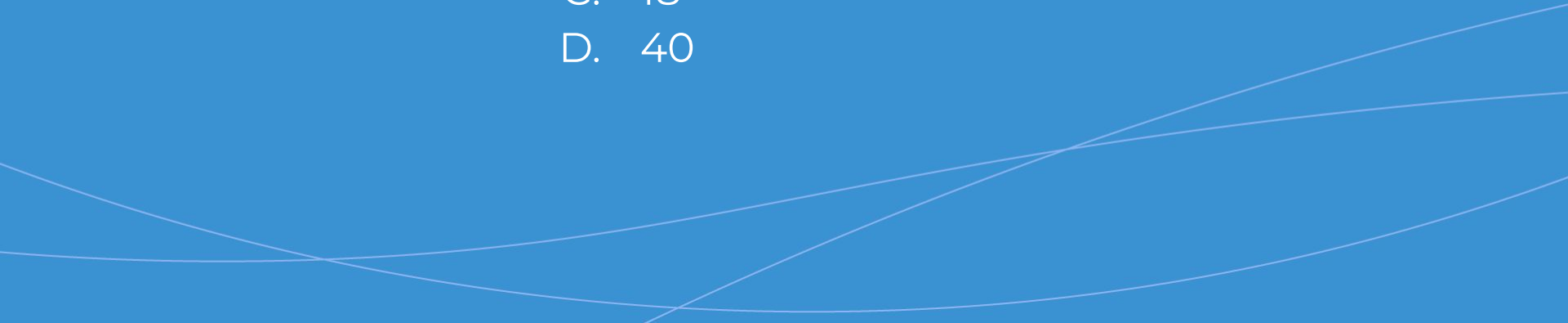


What will `len("Hello")` return if we store it as a variable?

- A. "Hello"
 - B. "5"
 - C. 5
 - D. 5.0
- 



What will `20%2` return if we store it in a variable?

- A. 0
 - B. 10
 - C. 18
 - D. 40
- 

Casting Data Types

- ★ In Python, we can convert variables into other data types should we need to. This is known as casting.
 - Cast to String → `str()`
 - Cast to Integer → `int()`
 - Cast to Float → `float()`

Booleans

- ★ Booleans can only store one of two values : True or False.
- ★ These are mainly used for conditional checks.
- ★ Booleans should be declared with capitals. Using lowercase for booleans will return an error in Python.
- ★ Example :

```
var = True
```

```
var2 = False
```

```
# Notice how 'true' and 'false' lights up a  
# different colour.
```

Integers & Floats as Booleans

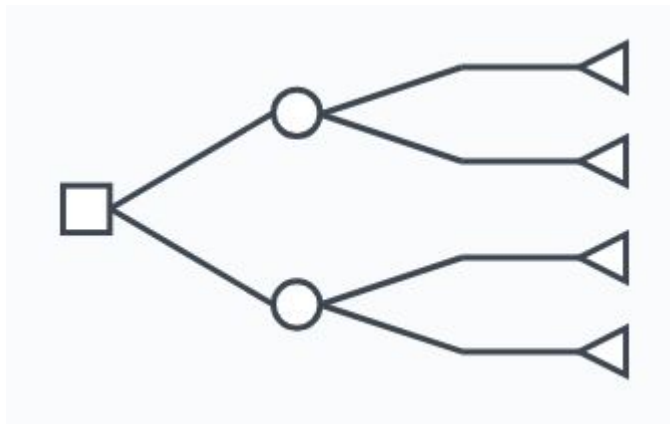
- ★ Both integers and floating point numbers can be converted to boolean using the `bool()` function.
- ★ An int, float, or complex number set to zero will return as `False`.
- ★ An int, float, or complex number set to any other value that is not zero, returns `True`.

Getting started with Control Structures

- ★ **Control structures are code that will analyse variables and then choose a direction to follow based on the provided input.**
- ★ **Think of it as a form of branching : depending on the provided input, your program will have one of x branches to follow.**
 - **E.g. “If I finish my work early, I will go to bed. Else, I will have to work through the night.”**

Illustration of branching control structures

★ Example from www.lucidchart.com



If Statement & Syntax

if <condition>:
 <statement>

```
x = 10
if x > 6:
    '''The condition within the if statement is
    true, therefore the below print command
    will execute'''
    print("x is greater than 6")
print("""This print is not within the scope of the if statement
therefore it will print regardless if the condition is true or false
""")
```

Else Statements

- ★ We now know that we can use if statements to control the flow of our programs.
- ★ What if we wanted and alternative outcome?
- ★ This is where the else statement comes in.
 - E.g. if it is raining, I shall bring my coat, else I shall leave my coat at home.

Else Example

```
is_raining = False
if is_raining == True:
    print("Bring a coat")
else :
    print("Leave coat at home")
```

Elif Statements

- ★ What if there was a situation where we could have multiple statements that are True?
- ★ This is where elif comes into play : Else if → elif
- ★ Elif statements are mainly used to handle the case when multiple True statements are present.
- ★ Note that you can have multiple elif statements in an if-else block.

Elif Statement Example

```
user_num = int(input("Please enter a number : "))  
if user_num == 0 :  
    print("Please enter a number that is not zero")  
elif user_num < 10 :  
    print("Your number is less than 10")  
elif user_num > 10 :  
    print("Your number is greater than 10")  
else:  
    print("Are you sure you have entered a number?")
```

Things to Note

- ★ **There is no limit to the number of elif statements one can have in an if-else block.**
- ★ **Only one final else statement is allowed, per block.**
- ★ **Each condition is checked in order.**
- ★ **If one condition is true, that branch executes, and the statement ends.**
- ★ **Even if there are multiple conditions that are True, only the first True branch will execute.**

Nested if Statements

```
grade = int(input("Enter your grade : "))  
if grade > 50:  
    if grade > 75:  
        print("You passed!")  
    else:  
        print("You passed, but you can do better!")  
else:  
    print("You failed!")
```


- ★ So far, we have used a few operators, namely:
 - Assignment (=)
 - Equal to (==)
 - Greater than (>)
 - Less than (<)
- ★ Here we will cover more operators available to us and how to utilise them.

Comparison Operators

OPERATOR	OPERATION	EXAMPLE
<code>==</code> Equal to	True if x has the same value as y	<code>x == y # True</code>
<code>!=</code> Not equal to	True if x does NOT have the same value as y	<code>x != y # False</code>
<code>>=</code> greater than or equal to	True if x is greater than or equal to y	<code>x >= y # True</code>
<code><=</code> Less than or equal to	True if x is less than or equal to y	<code>x <= y # True</code>

Logical Operators

OPERATOR	OPERATION	EXAMPLE
and	True if both x AND y are true (logical conjunction)	If x and y : print(z)
or	True if either x OR y are true (logical disjunction)	If x or y print(z)
not	True if the opposite of x is true (logical negation)	If not x print(y)

and Operator

- ★ Returns as True when both conditions specified are met.
- ★ Example:

```
if 10 < 50 and 500 > 100:  
    print<"This is a conjunction">  
else:  
    print<"Not a conjunction">
```

or Operator

- ★ Returns True if either of the specified conditions are met.
- ★ Example:

```
if 10 < 50 or 500 > 100:  
    print("This is a disjunction")  
else:  
    print("Not a disjunction")
```

not Operator

- ★ Changes the condition from True to False and vice versa.
- ★ Example:

```
if not 100 < 500:  
    print("This is negation")  
else:  
    print("Not negation")
```

What will the following condition do?

If true:

<code block>

- A. It will skip the code block
- B. Nothing
- C. It will run the code block
- D. It will repeat the code block

What will the following condition do?

```
If 50 > 100:  
    print("That's true!")
```

- A. It will display "That's true"
- B. Nothing
- C. It will add 50 to 100
- D. It will repeat the code 50 times

What will the following condition do?

If not false:
<code block>

- A. It will skip the code block
- B. Nothing
- C. It will run the code block
- D. It will repeat the code block

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**



CoGrammar

Thank you for joining!