Data Structures and Programmatic Thinking. Session 13

Pepe García

2020-04-20

Data Structures and Programmatic Thinking. Session 13

Plan for today

- Learn what's JSON
- See how it relates to Python data structures

Datatypes

Datatypes help us categorize our values into different *sets*

Datatypes

arithmetic operators

Operators are symbols in the language that perform different kinds of computations on values

They're binary

Arithmetic Operators

symbol	meaning
+	sum
-	substraction
	multiplication
/	division
	exponentiation
//	floored division
%	modulus

Boolean operators

We're going to learn two kinds of operators that operate on booleans Comparision and logical operators.

Boolean operations are useful for conditional execution.

Comparision operators

name	description\
x == y	x is equal to y
x != y	x is not equal to y
x > y	x is greater than y
x < y	x is lesser than y
x >= y	x is greater than or equal than y
x <= y	x is lesser than or equal than y
x is y	x is the same as y
x is not y	x is not the same as y

Logical operators

name	description
x and y	returns True if x and y are true
x or y	returns True if either x or y are true
not x	negates x

Variables

Variables are names that point to values in Python.

Naming rules

- variable names can't start with a number
- variable names can't contain special characters such as !, @, .
- Can't be one of the reserved words

Mutability

In Python variables are mutable. This means that we can change their value at any time

```
name = "Pepe"
print(name)


name = "Jose"
print(name)
```

Functions

Functions are sequences of instructions that we store to be executed later

Calling functions

The syntax for calling functions is the following:

```
function_name(parameter1, parameter2)
```

Declaring functions

We can declare our own functions using the def keyword with the following syntax:

```python
def function_name(parameter1, parameter2):
    #function body
```

If statement

the if statement is the tool we use for conditional execution in Python

```
if <condition>:
    <body>
```

Else clause

The else clause is executed when the condition is evaluated to false:

```
if <condition>:
    <block>
else:
    <block>
```

Elif clause

Elif clauses are used when there are more possibilities:

```
if <condition>:
    <block>
elif <condition>:
    <block>
else:
    <block>
```

Iteration

Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

While

```
while <condition>:
    <body>
```

Stopping a loop

You can always stop a loop using the break keyword

```python
while True:
    if im_bored_of_iterating:
        break
    else:
        print("i'm iterating!")
```

Constructing Lists

We construct lists with the brackets [] syntax:

```
[1, 2, 3, 4, 5]
["hello", "dolly"]
[]
[1, "hello", 2, "dolly", 3]
```

Accessing list elements

We use **square brackets** to access elements by their **index**.

indices in lists start by **0**, not 1.

```
words = ["hello", "dolly"]
words[0]
# "hello"
words[1]
# "dolly"
```

Updating elements in the list

To update an element inside the list, we use a syntax similar to the one for declaring variables, but using the brackets and the index we refer to.

```
numbers = [1,2,4]
numbers[2] = 3
print(numbers)
```

Appending elements to the list

To add a new element to the end of the list we use the append() method on it.

```
numbers = [1,2,3]
numbers.append(4)
print(numbers)
```

Inserting elements in the list

There's an alternative way of adding new elements to the list, and it's using the insert() method on it:

```
words = ["hello","my","friends"]
words.insert(2, "dear")
print(words)
```

The difference between this and append is that with insert we can choose where to put it by using the target index

Removing elements from the list

In order to remove an element from a list, we should use the .pop() method, and pass the index of the element we want to remove

```
words = ["hello","my","friend"]
words.pop(1)
print(words)
```

For loops

```
for <variable_name> in <list>:
    <body>
```

Creating dictionaries

We use curly brackets ({}) to declare dictionaries.

```
translations = {
    "es": "Hola!",
    "it": "Ciao!",
    "en": "Hello!"
}
```

colon for separating key and value

comma for separating entries

Adding elements

We add elements to dictionaries given their specific index:

```
translations = {}
translations["en"] = "Hello"
translations["it"] = "Ciao"
translations["es"] = "Hola"
```

Updating elements

we always can change a value in the dictionary by re-assigning the key

```
translations = {}
translations["en"] = "Hello"
translations["en"] = "WHATUP!"
```

Deleting elements

We can delete an element of the dictionary using the **pop** method

```
translations = {}
translations["en"] = "Hello"
translations.pop("en")
```

Getting all keys or values

We can allways get all **keys** or **values** from the dict as a list using either the **.keys()** or **.values()** method

```
users = {
  1: "Pepe",
  22: "Peter",
  44143: "Johnny",
  2: "Chuck"
}

users.keys()
users.values()
```

Reading files

```python
with open("file_path") as file:

    for line in file:
        #do something with line
        print(line)
```

Writing files

We can write to files using a similar approach

```python
with open("file.txt", "w") as f:
    f.write("this content will be written to the file!")
```

Handling files. modes

When opening a file, we can choose in which **mode** we open it

CSV files

Python comes with a **CSV** library that we can use out of the box. We use it by **importing** it. **Imports** are commonly added at the top of the file.

```
import csv
```

CSV files

The **csv** library is based on the idea of readers and writers. One can read all lines in a file like so:

```python
with open("file.csv") as f:
    reader = csv.reader(f)
    for line in reader:
        print(line)  #line will be a list here
```

first we open the file normally

Then we create a reader using **csv.reader()**

Finally, we operate with the reader

CSV files

writing is not very different from reading:

```
lines = [
  ["asdf", "qwer"],
  ["hello", "world"]
]

with open("file.csv", "a") as f:
    writer = csv.writer(f)
    for line in lines:
        writer.writerow(line)
```

First we need some data to put in the csv file

Then we open the file with the append mode

Later, we create a **csv.writer**

Converting to JSON

```
import json
```

we can use the JSON library in Python by **importing** it

JSON

```python
ramones = [
  "Johnny",
  "Joey",
  "Markee",
  "Dee-dee"
]

# dumps function returns a json formatted string
json.dumps(ramones)
```

We can convert from Python data into a JSON string with **json.dumps** function

JSON

```python
ramones_as_a_string = '["Johnny", "Joey", "Markee", "Dee-dee"]'


# json.loads will convert from a json encoded string
# into Python data structures
ramones = json.loads(ramones_as_a_string)
```

We can convert from a JSON encoded string to Python data with **json.loads** function