Python Basics

Zeina Elguindi

2024

Contents

1 Basic Data Types

1.1 Overview

Class	Description
bool	Boolean value
int	Integer value
float	Floating point number
str	Text String
list	Mutable sequence of objects
tuple	Immutable sequence of objects
dict	Dictionary

1.2 Bool

The bool class is used to manipulate logic. There are only 2 instances of this class; True or False.

Input	bool Return
0 or 0.0	False
number $\neq$ 0 or 0.0	True
”” or ”	False
empty list/tuple	False
non-empty list/tuple	True

1.3 Strings

1.3.1 String Concatenation & Replication

String **concatenation** joins 2 or more strings together:

$$"Zeina" + "Elguindi" \rightarrow "ZeinaElguindi"$$

String **replication** duplicates a string 2 or more times:

$$"ZeinaElguindi" * 2 \rightarrow "ZeinaElguindiZeinaElguindi"$$

1.3.2 String Formatting

Escape Character	Ouput
\’	Single quotation mark
\”	Double quotation mark
\n	New line
\\	Single backslash

Alternatively, a raw string will ignore all escape characters:

$$print(r'ZeinaElguindi\backslash's') \rightarrow \ 'ZeinaElguindi\backslash's'$$

1.3.3 String Manipulation: Indexing & Slicing

String Indexing

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

alpha = hello

alpha[0] →'h'

alpha[4] → 'o'

alpha[-5] → 'h'

String Slicing
String slicing allows you to generate a sub string from any given string.

$$str[start : stop : step]$$

start: inclusive   stop: exclusive   step: optional

alpha[:5] → 'hello'

alpha[1:4] → 'ell'

alpha[-2:] → 'lo'

alpha[:] →' hello'

1.3.4 String Methods

Basic Methods	Descriptions
.isupper()	Converts string to uppercase
.islower()	Converts string to lowercase
.isalpha()	Returns True if string consists of letters only
.isdecimal()	Returns True if string contains numbers only
.isalphanum()	Returns True if string only contains numbers and letters
.strip()	Returns a string with without leading/trailing whitespaces

.join() Method
The .join() method is passed a list of strings which it will join using the string it is called upon.

$$' \ '.join([' alpha',' beta',' gamma']) \rightarrow \ 'alpha\ beta\ gamma'$$

$$',\ '.join([' alpha',' beta',' gamma']) \rightarrow \ 'alpha,\ beta,\ gamma$$

.split() Method
The .split() method is passed a string value, which it will use to split up the string it is called upon. Essentially the opposite of .join() method.

('Zeina Elguindi').split() → ['Zeina', 'Elguindi']

('zeina elguindi').split('e') → ['z','eina','elguindi']

1.4 Lists

Lists are mutable sequences of objects, meaning the list elements can be changed. A list can contain more lists within it, and a list can contain a combination of different data types.

1.4.1 List Concatenation & Replication

$$[1, 2, 3] + ['a', 'b'] \rightarrow \ [1, 2, 3, a, b]$$

$$[1, 2, 3] * 2 \rightarrow \ [1, 2, 3, 1, 2, 3]$$

1.4.2 List Manipulation: Indexing & Slicing

List Indexing

['cat',	'dog',	'mouse']
0	1	2
-3	-2	-1

$$animals = ['cat', 'dog', 'mouse']$$

$$animals[0] \rightarrow \ 'cat'$$

$$animals[-2] \rightarrow \ 'dog'$$

$$animals[-5] \rightarrow \ error$$

$$print('Hi' + animals[1]) \rightarrow \ 'Hi\ cat'$$

List Slicing

$$animals[:2] \rightarrow \ ['cat', 'dog']$$

$$animals[0:-2] \rightarrow \ ['cat']$$

$$animals[-2:] \rightarrow \ ['dog']$$

$$animals[:] \rightarrow \ ['cat', 'dog', 'mouse']$$

1.4.3 Element Assignment

You can manipulate single list elements by doing the following:

$$animals[0] =' hamster' \rightarrow \ animals = ['hamster', 'dog', 'mouse'$$

$$animals[1] = animals[0] \rightarrow \ animals = ['hamster', 'hamster', 'mouse']$$

Multiple Variable Assignment You can assign a variable to each element in the list by doing the following:

$$course = ['MSE111', 'D.Walter', 'Medium']$$

$$class, professor, difficulty = course$$

Note: The number of variables must be equal to the length of the list (len(course) = 3, therefore 3 variables must be assigned)

1.4.4 List Methods

$$animals = ['cat','dog','mouse']$$

.index(list item) returns the index value of the element within the list. If there are duplicate elements, only the index of the first element will be returned.

- animals.index('dog') returns 1

.append(new item) appends a new element to the end of the list Note: If you want to choose where the element gets added, use **.insert(index, new item)**.

- animals.append('t-rex') would mean animals = ['cat','dog','mouse','t-rex']

These methods modify the list in place, therefore, you do not need to re-assign the return value.

.remove(list item) removes the passed in list item

- animals.remove('t-rex') would mean animals = ['cat','dog','mouse']

.sort() sorts the list alphabetically or numerically

- .sort() sorts the list in an ascending order
- .sort(reverse=True) sorts the list in a descending order

Sort method cannot be used on lists containing elements with different data types.

1.5 Dictionaries

Dictionaries are similar to lists as they can store many values with a variety of data types. In a dictionary, the 'index' is called a 'key' and it can be of any data type, unlike a list's index. The value associated with the key is called a 'value' and together, they are the *key-value pair.*

$$birthdays = \{'Jack' :' Apr1','Ben' :' May21','Will' : Nov3\}$$

In this example, the name is the key/identifier (it replaces the integer index as exists with lists), and the birthday is value associated to the key.

$$birthdays['Ben'] \rightarrow\ 'May21'$$

To set a new pair in the dictionary, set a value to a new key. The following example adds Frank and his birthday to the dictionary:

$$birthdays['Frank'] \rightarrow\ 'Jan8'$$

1.5.1 Dictionary Methods

dict.keys() returns all the keys in the dictionary. Output data type: tuple.
dict.values() returns all the values in the dictionary. Output data type: tuple.
dict.items() returns all entries in the dictionary. Each pair is represented as a tuple within a list.
dict.get(key,fallback) returns the value associated to the key, if it exists. If it doesn't exist, the fallback value is returned.

2 Functions

2.1 Function Parameters

- **Argument** is the value that you pass into a function
- **Parameter** is the variable that the argument is stored in

Default Values A function's arguments can have default values. You use the "=" operator to assign a default value to an argument, which makes passing that parameter to the function optional.

- If a function is called without an argument, then the default argument is used
- If a function accepts 3 parameters, and of those, 2 are default, then 1 argument must be passed in, and the other 2 are optional.
  - If you wish to pass in optional arguments, you must use a keyword to specify which parameter the argument belongs to.
    Note: This is not necessary if you are passing in arguments in the order that they are accepted in.

Once a function returns, the value stored in the parameter is forgotten.
The print() function also contains optional parameters 'end' and 'sep' to specify what should be printed after the argument:

- print() adds a new line character at the end of it's argument, but this can be bypassed by including passing in an argument $end = ''$.
- You can use the $sep = ','$ argument to separate the strings passed into the print function.

2.2 Return Values

The return statement is the final and terminating execution of a function

- If a value is returned, this value is passed back to the orginal function call statement
- If no value is specified for return, the 'None' value is returned

2.3 Local and Global Scope

Parameters and variables assigned in a function are in that function's*local scope*, such variable is called a *local variable*. Variables assigned outside all functions exist in the *global scope*.

- As mentioned previously, when a function returns, all variables assigned within it (local variables) are forgotten.
- Variables in different scopes can have the same name as they are able to store different values within their scopes.

Global variables can be used in the local scope, although the reverse is not true.
If you would like to use/alter a global variable within a local scope, declare the global statement: $globalvarname$ within the function. The program will now know that varname refers to the global variable, and there is no need to create a new local variable with that name.

2.4 Packing and Unpacking Sequences

Automatic Packing
If a series is comma-separated, then it automatically gets formatted to a tuple when assigned.

$$even = 2, 4, 6 \iff even = (2, 4, 6)$$

Although you cannot have more than one return statement in a function (unless through the use of logic statements), you can return more than one value within the same return statement:

$$return\ x, y$$

This will return a tuple with the values of x and y as items.

Automatic Unpacking
Python can also automatically unpack iterable objects (lists, tuples, etc).

$$even = 2, 4, 6$$

This can also be used with loops:

```
for x,y in [(1,3), (7,8)]
for k,v in animals.items()
```

The first loop will assign x and y to each value within the tuple. The second loop will iterate over the key and value of the animals dictionary items.

2.4.1 Simultaneous Assignment

$$x, y, z = [1, 2, 3]$$

$$x = 1$$

$$y = 2$$

$$z = 3$$

This feature is especially useful for variable swapping:

$$x, y = y, x$$

The variable 'x' will be assigned to the old value of y, and the variable 'y' will be assigned to the old value of x.
This swapping method is more efficient than the following:

$$temp = x$$

$$x = y$$

$$y = temp$$

3 Classes and Instantiation

3.1 Classes Overview

A class serves as a user-defined template used to create objects. A class has defined attributes and methods (functions) that its created objects will have.

- Provides a set of behaviors for instances of the class (objects) to follow
- Defines the properties of the class's objects

3.2 Components of a Class

Attributes
Attributes are variables/properties of the class or its instances.
For example, a 'Student' class may have attributes of name, year and program

```
student.name = Jack Hemington
student.year = 4
student.program = Economics
```

Methods
Methods are defined functions/behaviors that objects in the class can perform. For example, the 'Student' class may have a method called study, which makes the student study.

```
def study(self):
    print(f'{self.name} is now studying!')
```

Constructor Method
The constructor method instantiates an object, a.k.a, it initializes the attributes of the class.
When a new instance of the class (object) is created, the constructor method is automatically called to assign the arguments and return an initialized object.

```
class Student:
def __init__(self, name, year, program):
    self.name = name
    self.year = year
    self.program = program

student1 = Student('Jack Hemington', 4, 'Econonomics')
```

3.3 Creating a Class

Defining a Class

- Define the class
- Set up the constructor method and attributes
- Define a method for the class

Example:

```
class Student:
    def __init__(self, name, year, program):
    self.name = name
    self.year = year
    self.program = program

    def study(self):
        print(f'{self.name} is now studying!')
```

Instantiating a Class To create an object/instance of the class, you call the class as if it were a function, with the necessary values to pass in. Example:

```
student1 = Student('Jack Hemington', '4', 'Economics')
```

The constructor method will then be called to initialize this object.

Accessing Attributes and Methods
Accessing Attributes:

objectName.attributeName

Accessing Methods:

objectName.methodName

3.4 Complete Example:

```
class Student:  # define the class

    def __init__(self, name, year, program):  # define constructor method
        self.name = name
        self.year = year
        self.program = program

    def study(self):   # define method
        return f'{self.name} is now studying!' # access attribute

# create instances of the 'Student' class
student1 = Student("Andy", 4, "Arts and Literature")
student2 = Student("Ben", 3, "Marketing")

# call method and access attributes
print(student1.study())   # calls 'study' method
print(student1.program)   # output: "Arts and Literature"
print(student2.year)      # output: 3
```

3.5 Purpose of ”Self”

The self parameter is used to access variables that belong to the class.

- It must be the first parameter in any methods defined within the class.

4 Exception Handling

4.1 Raising an Exception

You can raise exceptions to validate user input or parameter types.
Checking the validity of a parameter for a division function may look like the following:

```
def divide(dividend, divisor):
    if divisor == 0:
        raise ValueError("Divisor cannot be zero.")
    return dividend / divisor
```

When an exception is raised, a new instance of the Error class is created, with the error message serving as a parameter for the constructor (chapter: 3.2 for more details).
You can raise built-in exceptions such as 'ValueError' or 'TypeError', or you can raise an exception from a custom exception class.

```
class CustomError(message):
    def __init__(self, message):   # constructor method
        self.message = message
        super().__init__(self.message)  # calls constructor of Exception class
def sqrt(x):
    if x < 0:
        raise CustomerError ('x must be greater than 0.')
try:
    print(sqrt(-3))
except CustomError as error:
    print(f'Error: {error}
```

4.2 Catching an Exception

To avoid the chances of an exception being raised, put code that may have an error in a *try* clause. Once an error is detected, the program will move to the *except* clause.

- The program will move to the except clause once an error is raise, and it will not return to the try clause.

```
try:
    file = open('sample_file.txt')
except IOError:
    print('Invalid file path')
```

4.2.1 Example:

```
def divide(divisor):
    try:
        return 30/divisor
    except ZeroDivisionError:
        return 'Division by 0 is not possible.'
print(divide(3))  # output: 10
print(divide(0))  # output: 'Division by 0 is not possible.'
print(divide(10)) # output: 3
```

Note: In this case, the final line gets executed, although if the *try* clause was surrounding the print statements, '3' would not be outputted as the program will move to the *except* clause and not return.

5 Text File Handling

5.1 Preparation

File Path
When defining file paths from a location on your computer, you must double your backslashes, as each backslash must be escaped by another.

$$filepath =' C : WindowsUsersJohnword.docx'$$

Opening and Closing Files

filename.open() or filename.close()
with open(filename, ”x”) s.t. x = r(read), w(write), or a(append)

If the filename does not exist, the program will create a new and blank file. **Example File Contents**

In the following sections, I will be referring to a file called ’numbers.txt’ which has the following contents:

```
The following is a list of numbers:
1
1.5
3.0

-5
```

5.2 Reading from a File

Starter File Reading Methods

Methods	Descriptions
.read()	Returns contents of file as a sting
.readline()	Returns current line of a readable file as a string
.readlines()	Returns all lines of a readable file as a list of strings

Example

```
def read_numbers(filename):
    with open(filename, 'r') as infile:
        infile.readline()   # read header line
        numbers_list = []

        for line in infile:
            if line.strip() != "":
                numbers_list.append(line.strip())
    return numbers_list

print(read_numbers('numbers.txt'))
```

Note: The method .strip() was used to strip each line from new line characters, as well as leading and trailing white spaces.

Output

```
[’1’, ’1.5’, ’3.0’, ’-5’]
```

5.3 Writing to a File

When you write to a file, may content originally existing within the file gets overwritten. If this is not the intended outcome, you can append to the file by passing 'a' in the open statement, or you can determine the last line in the file and begin writing from there. It is also important to note that when writing to a file, newline characters must be explicitly included in the string statements.

5.3.1 Overwriting Example:

```
numbers_list = [10, -5.5, 3.14]
filename = 'numbers.txt'

with open(filename, 'w') as outfile:
    for number in numbers_list:
        outfile.write(f'{number}\n')  # add newline character after each number
```

Output:

```
10
-5.5
3.14
10000000000.0
```

5.3.2 Appending Example:

```
numbers_list = [10, -5.5]
filename = 'numbers.txt'

with open(filename, 'a') as outfile:
    for number in numbers_list:
        outfile.write(f'{number}\n')
```

Output:
1
1.5
3.0

-5
10
-5.5

5.3.3 Using 'w' Without Overwriting

You have to read the file contents, and re-write them as well as write your desired additional text.

```
    numbers_list = [10, -5.5]
    filename = 'numbers.txt'

    with open(filename, 'r') as infile:
        original_content = infile.readlines()
    with open(filename, 'w') as outfile:
        outfile.writelines(original_content)
        for number in numbers_list:
            outfile.write(f'{number}\n')
```

Output: Same as the example above.

6 Pandas Library

7 Series

A series is a one-dimensional array.

- A series is essentially a column in a spreadsheet
- The series axis is the *index*

7.1 Creating a Series

series = pd.Series(data, index, dtype)

Argument	Type	Description
data	list / tuple	Data to be displayed within the column.
index *(optional)*	list / tuple	Index labels. Default: Integer Indexing.
dtype *(optional)*	data type	Data type specification. Default: Inferred.

7.2 Accessing Elements in a Series

To access an element(s), use the [ ] operator, with the index (type: int) that you would like to access.

Example 1:

```
data = [1,2,3]
ser = pd.Series(data)
print(ser[2])
Output: 3
```

Example 2:

```
print(ser[:1])
Output:
 0 | 1
 1 | 2
```

7.3 Indexing and Selecting Data

Operator/Function	Name	Purpose
[ ]	Indexing Operator	
.loc[]	Label Location	Selecting data by index label
.iloc[]	Integer Location	Selecting data by integer index

Note: Using series, there are many similarities between these 3. .loc[] uses the index names to select data and is therefore inclusive, while .iloc[] uses index positions, and is therefore end-exclusive.

8 DataFrames

A dataframe is a two-dimensional data structure. The datasets within it are stored in 2 axes, the rows and columns (essentially a table)

8.1

Creating a DataFrame

8.1.1 From an aaray

```
arr = ['Physics', 'Chemistry', 'Biology']
df = pd.DataFrame(arr)
```

The output of this would be a table, with one column. Each row represents the different list elements, and the index and column labels represent the row and column headers respectively.

8.1.2 From a Dictionary

```
hi
```

8.2 Viewing Data

8.3 Selecting Data