NOVA IMS Information Management School

Data Mining

Introduction to Python

Ivo Gonçalves
igoncalves@novaims.unl.pt

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Python

- High-level programming language

- Dynamic typing, i.e., variable types are inferred during program execution

Python vs. Java

- Some initial differences are:
  - Python does not have variable declarations
  - In Python no semicolons (“;”) are needed to finish statements
  - Python uses indentation instead of braces (“{“ and “}”) to define code blocks

Main types

- **int:** integers
- **float**: floating-point numbers
  - The double type does not exist
- **bool**: boolean
  - **True** or **False** (notice the first capital letter)
- **str**: strings
  - immutable type as in Java
- **Lists**: dynamic arrays

Creating integers, floats, and booleans

- int:

```
a = 42
b = 100
```

- float:

```
a = 42.0
b = 100.55
```

- bool:

```
a = True
b = False
```

Creating strings

- Strings can be defined within double quotes as in Java, or within single quotes

- For example:

  s1 = “Python”

  s2 = ‘Python’

- The **len** function can be used to get the length of a string:

  len(s1)

  len(s2)

Arithmetic Operators

- + Additive operator
- - Subtraction operator

- * Multiplication operator
- / Division operator
- % Remainder operator

Division operator behavior with integers

- Python and Java have different behaviors when handling division between two integers (**a** and **b**)
- In Python, **a / b** returns the real-valued division (i.e., with the corresponding fractional part)
  - In the following example **a** is set to 0.5:

```
a = 1 / 2
```

- In Java, **a / b** returns the integer division (i.e., without the corresponding fractional part)
  - In the following example **a** is set to 0:

```
int a = 1 / 2;
```

- To perform an integer division in Python use the **//** operator
  - In the following example **a** is set to 0:

```
a = 1 // 2
```

++ and -- do not exist

- **[variable]++** or **[variable]--** are not valid statements in Python
  - For example, the following does not work:

```
a = 5
a++
```

- Use **[variable] += 1** instead of **[variable]++**, and **[variable] -= 1** instead of **[variable]--**
  - For example:

```
a = 5
a += 1
```

Some useful math functions

- In Python, the **math** module starts with a lower case letter

- To compute the square root of ***a**:*
  math.sqrt(a)

- To compute ***a** raised to the power of **b**:*
  math.pow(a, b)

  or,

  *a ** b*

- To get the value of ***π**:*
  math.pi

Importing modules/packages

- As in Java, the **import** keyword is used to import/include a given module/package

- In Python, **math** is one of the modules that needs to be imported:

  import math

Importing modules/packages

- Local module names can be assigned when importing a module by writing:

  import [module_name] as [local_module_name]

- For example, instead of writing:

  ```
  import math
  math.sqrt(25)
  ```

- You could write:

  ```
  import math as m
  m.sqrt(25)
  ```

Relational operators

- **==** **equal to** (also works for strings)
- **!=** **not equal to** (also works for strings)

- **<** **less than**
- **>** **greater than**

- **<=** **less than or equal to**
- **>=** **greater than or equal to**

Control flow statements

- In Python, the **if/else if/else** statements are similar to Java
- However, instead of “**else if**”, Python uses “**elif**”
- It also uses indentation to define where the blocks start and end

Control flow statements

- General structure (notice the colons and the lack of braces):

```
if ([expression]):
        [statement(s)]

elif ([expression]):
        [statement(s)]

else:
        [statement(s)]
```

Logical operators

- **and** is the logical AND operator in Python
  - Java uses &&
- **or** is the logical OR operator in Python
  - Java uses ||

Lists

- A list is a dynamic array, i.e., an array that can increase or decrease in size as needed

- In Python, a list can contain variables of different types

- Creation of an empty list:

```
l = []

or,

l = list()
```

Indexing

- Lists and strings can be indexed as commonly done in an array

- If a list or a string has size N, then the valid indexes are between 0 and N – 1

- The length of a list can be obtained by using the **len** function as in the string case:

  len(l)

Indexing

- Besides the standard indexation between 0 and N – 1, Python also allows to start indexing from the end of the list or string

- Indexing a list or string in the index -1, returns the last element of the list or string
  - For example, the following code returns the last element of list l:

    ```
    l[-1]
    ```

- Following the same reasoning, indexing a list or string in the index -2, returns the second element of the list or string counting from the end

Slicing

- Slicing is one of the most powerful features of Python
- Slicing consists of returning a copy of a particular area of a list or string
- There are several slicing possibilities

Slicing

- 1) **listX[:],** returns a copy of all the elements of listX

- 2) **listX[a :],** returns a copy of all the elements of listX starting at index **a** and until the end

- 3) **listX[: b],** returns a copy of all the elements of listX starting at the beginning and until index **b - 1**

Slicing

- 4) **listX[a : b],** returns a copy of all the elements of listX starting at index **a** and until index **b - 1**
- 5) **listX[a : b : c],** returns a copy of all the elements of listX starting at index **a** and until index **b – 1**, but jumping **c** values at each time
  - For example:

    listX[0 : 5 : 2], visits index 0, 2, and 4
- 6) **listX[a : b : -c],** returns a copy of all the elements of listX starting at index **a** and until index **b – 1**, but jumping -**c** values at each time
  - For example:

    listX[5 : 0 : -2], visits index 5, 3, and 1

Slicing

- 7) **listX[: : -1],** returns a copy of all the elements of listX in reverse order

Functions

- General structure:

```
def <function_name>(<list of parameters>):
    [statements]
```

For loop

- In Python there are two main ways of constructing a for

- 1) Controlling the iterations with an explicit number

```
for i in range(n):
        [statements]
```

For loop

- 2) Explicitly iterating over a collection (list or string):

```
listX = [5, 10, 15]
for i in listX:
        [statements]
```

List comprehensions

- List comprehensions are a very compact way of creating lists

- General structure:

listX = [<expression> for i in range(n)]

Random numbers

- Random numbers can be used with the **random** module:

  import random

- random.random(), generates a float between [0, 1[

- random.randint(a, b), generates an integer between [a, b]

Exercise

- Create a Python program that generates N random 2D points with coordinates between [0, 100[
- Each point should be assigned to a random cluster among K possible clusters
- Save the results in a list
- Print the output