Statistical Computing

Gabriele Rovigatti

Bank of Italy

March 5, 2021

UNIVERSITA' degli STUDI di ROMA
TOR VERGATA

Today

- **Python basics**
  - General Framework
  - Data Types
  - **Syntax**
  - **for loops, if/else**
  - **Modules / Libraries**
  - **Functions**
- Webscraping
  - Data Storage
  - HTML syntax
  - `Requests` package
  - `Webdriver` package
- Text Mining?

Working with Dictionaries

Elements in dictionaries are called by either *['key']* or by the *get()* method. The latter returns *None* instead of *KeyError* if the key is not in the dictionary

```
# Initiate a dictionary
lessons = {
1 : 'Basics',
2 : 'Syntax',
3 : 'Webscraping'
}
lessons[1] # 'Basics'
lessons[1] = 'Basics + Syntax Elements' # CHANGE item
lessons.get(1) # 'Basics + Syntax Elements'
lessons[116] = 'Machine Learning' # ADD item
print(lessons) # {1: 'Basics + Syntax', 2: 'Syntax', 3: 'Webscraping', 116:
lessons.get(115) # NULL
lesson[115] # KeyError: 115
```

Working with Dictionaries / 2

```
# length of a dictionary is the number of "key : value" pairs
len(lessons) # 4
# 'in' is used to check whether a key exists in a dictionary (returns a boo
4 in lessons # True
# update() method adds new "key : value" pairs to the dictionary
additional_lessons = {4 : 'Statistical Computing', 5 : 'Research'}
lessons.update(additional_lessons)
# pop() method: removes the item and returns the value
lessons.pop(2) # 'Syntax'
# del: removes the item
del(lessons[3])
# .values(), .keys() and items() methods: return values, keys and tuples as
lessons.items() # dict_items([(1, 'Basics + Syntax Elements'), (116, 'Machi
lessons.values() # dict_values(['Basics + Syntax Elements', 'Machine Learni
lessons.keys() # dict_keys([1, 116, 5, 4])
# clear() method deletes all items at once
lessons.clear()
# del: removes the dictionary itself
del(lessons)
```

Many ways of coding - same outcome

Beginner

```
print("""There are 2 things to do in order to start working with Python:
1) download and install it at "http://www.python.it/download/";
2) follow closely a very good tutorial""")
```

Intermediate

```
print("""There are %g things to do in order to start working with Python: \n
%g) download and install it at "http://www.python.it/download/"; \n
%g) follow closely a very good tutorial.""" % (2,1,2))
```

Advanced

```
def printer(steps):
        p = range(steps)[1::]
        return(max(p),min(p),p[1])
if __name__ == '__main__':
        print("""There are %g things to do in order to start working with Python: \n
        %g) download and install it at "http://www.python.it/download/";\n
        %g) follow closely a very good tutorial. """ % printer(3))
```

Beginner Intermediate Advanced tryout

Indentation - when blanks do matter

In Python, leading blanks (*indentation*) are crucial in loops!

Indentation

```
for i in range(3):
print('This is invalid syntax!') #after a 'for','if',ect. Python expects at least one indentend command

# blanks matter, but not everywhere
b0=1
b1 = 1
b2  =  1
print(b0, b1, b2) #these are all equivalent
b = list() #initiate an empty list, equivalent to b = []
for i in range(3):
    b.append(i)
    print('We are INSIDE the loop, hence will print ' + str(b)) #it will print the list b at each iteration
print('We are OUTSIDE the loop, hence correctly prints ' + str(b)) #it will print b once
```

Run

For loops

The Python for statement iterates over the members of a sequence - or numbers, or a string - in order, executing a block of code each time.

```python
for iterating_var in sequence:
statement(s)
```

The first item in the sequence is assigned to the iterating variable `iterating_var`. Next, the statements block is executed. Each item in the list is assigned to `iterating_var`, and the statement(s) block is executed until the entire sequence is exhausted.

For Loops - Examples

```python
for letter in 'Python':      # First Example
   print('Current Letter :', letter)

fruits = ['banana', 'apple',  'mango']
for fruit in fruits:         # Second Example
   print('Current fruit :', fruit)

print("Good bye!")
```

if..elif..else

An else statement can be combined with an if statement. An else statement contains the block of code that executes if the conditional expression in the if statement resolves to 0 or a FALSE value.
The else statement is an optional statement and there could be at most only one else statement following if.

```
if expression:
   statement(s)
else:
   statement(s)
```

Dictionary Comprehension

- is a concise way to generate a dictionary from an iterable object;
- consists of a (key: value) pair followed by *for* statement

```
# generate the square of the fibonacci series
fibonacci_list = [1,1,2,3,5,8,13] # ITERABLE object
fibonacci_sq_dict = {x : x*x for x in fibonacci_list}
fibonacci_sq_dict # {1: 1, 2: 4, 3: 9, 5: 25, 8: 64, 13: 169} *** look at t
# the equivalent for loop reads
fibonacci_sq_dict_long = {} # initiate the dict
for x in fibonacci_list:
fibonacci_sq_dict_long[x] = x*x
# we may want to add more if and for statements
fibonacci_sq_dict_odd = {x : x*x for x in fibonacci_list if x%2 == 1}
fibonacci_sq_dict_odd
```

Exercises

- Write a Python script to concatenate following dictionaries to create a new one solution
    - dict1 = {1:10, 2:20}
    - dict2 = {3:30, 4:40}
    - dict3 = {5:50, 6:60}
- Write a Python script to generate and print a dictionary that contains the square of numbers between 1 and n (`n=int(input("Input a number "))` allows you to input integers). n = 10 solution
- Write a Python program to sum all the items in the previous dictionary solution
- Write a Python program to map two lists into a dictionary solution
    - colors = ['red', 'green', 'blue']
    - value = ['#FF0000','#008000', '#0000FF']
- Write a Python program to store all elements of an iterable (e.g., the string "python for webscraping") as keys, and as values the number of times they appear. solution

“Bonus” exercise: the Fibonacci Series

Write a Python script to generate and print a list that contains the Fibonacci series of numbers between 1 and n, where n is given as input when the code starts. Features of Fibonacci series:

1. the first two numbers of the series are [1,1] and are predetermined;[1]
2. the sequence F_n is defined by the recurrence equation: $F_n = F_{n-1} + F_{n-2}$

The solution is straightforward: we follow the rule with an exception!

[1]Some use [0,1], it is a matter of definition.

Fibonacci Series: (One of the) Solutions

```
import sys # not necessary, just to print the error
n = int(input("Input a number greater than 2 ")) # read the input n
if n <= 2: # check whether n > 2. If it is not the case, throw an e
    raise ValueError('You must input n > 2!')
    sys.exit()
# for loop solution
d1 = [1, 1] # initiate the list with the predetermined elements
for x in range(2,n+1):
    d1.append(d1[x-1] + d1[x-2]) # append the sum of previous two va
print(d1) # print the resulting list
```

Arithmetic and Comparison Operators

```python
# numeric objects
a = 10 # 10
a += 1 # 11 --
a -= 1 # 10
a % 3 # 1 -- modulo operator:
a ** 2 # 100

# string objects
animals = 'Cat ' + 'Dog '
animals += 'Monkey' # 'Cat Dog Monkey'
animals_list = ['Cat', 'Dog', 'Monkey']
' and '.join(animals_list) # 'Cat and Dog and Monkey'

# Arithmetic Comparison
a > 10 # False
a >= 10 # True
a == 10 # True
a != 10 # False
```

Comparison

```python
# Logical
A and B
A or B
not A
(A and (B or C))

# Identity Comparison
1 is 1 # True
1 is not '1' # True
bool(1) # Boolean Logical
bool(True) # This works with nearly every object
True is bool(1) # True
```

Functions

Functions in Python are defined according to simple rules:

- The function block begins with **def** followed by `functname` and parameters within brackets
- The code block within every function starts with a colon (:) and is indented
- The statement `return(object)` exits a function, optionally passing back an object to the caller

Check whether input is string

```
def str_check(parlist): # here we define the function: def, name, list parameters
    array = list(isinstance(x, str) for x in parlist) #this is indented!
    return(array)

print(str_check(['this',2,'string'])) # call the function with a parameter list
```

Fibonacci Series: Define a Function

```
import sys # not necessary, just to print the error
### FUNCTION DEFINITION ###s
def fibonacci_series(series_length, double_ones=True):
    if series_length <= 2: # check whether n > 2. If it is not the case, throw an er
        raise ValueError('You must input n > 2!')
        sys.exit()
    # for loop solution
    if double_ones == True:
        fibonacci_list = [1, 1] # initiate the list with the predetermined elements
    else:
        fibonacci_list = [0,1]
    for x in range(2,series_length+1):
        fibonacci_list.append(fibonacci_list[x-1] + fibonacci_list[x-2])
        # append the sum of previous two values to the list
    return fibonacci_list # return the list as output of the function
### ROUTINE ###
n = int(input("Input a number greater than 2 ")) # read the input number
d1 = fibonacci_series(n)
print(d1) # print the resulting list
d2 = fibonacci_series(n,0)
print(d2)
```

Import Modules

```
          # Imports the datetime module into the current namespace
import datetime
datetime.date.today()
# Import datetime and adds date and timedelta to the current namespace
from datetime import date, timedelta
date.today()
# Rename imports
import date as my_date
# This is sometimes used, but it is not suggested
from datetime import *
```

Exercises - **Mandatory**

- Write a function called **CipCiop** that takes a number and
  - If the number is divisible by 3, it should return "Cip";
  - If it is divisible by 5, it should return "Ciop";
  - If it is divisible by both 3 and 5, it should return "CipCiop";
  - Otherwise, it should return the original number.
- Write a function that takes as input an integer (called *limit*) and prints all the prime numbers between 0 and limit.
- Program a Rock-Paper-Scissors game, with the input() method (twice!), comparing the plays and printing the winner. (Yes, it is the same exercise for which you have the solutions, I will check your code. Try not to copy!)

Exercises - **Additional**

- Write a program that takes two lists as input and returns the common elements between the two lists. Make sure your program works on two lists of different sizes. Solution
- Write a one-line python code that takes a numeric list as input and returns the even elements of the list; in **one** line. Solution