Statistical Computing

Gabriele Rovigatti

Bank of Italy

March 4, 2021

This Course

- 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?

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?

Best friends - Error Messages

In case of error, every interpreter/software will try to tell you what is wrong in order for you to fix it.

Objects and errors

```
string_number = '1'
numeric_number = 1
intro = 'You can contatenate a string with a '
print(intro + str(type(string_number)) + ' object like ' + string_number)
print(intro + str(type(numeric_number)) + ' object like ' + str(numeric_number) + ' using str()')
print(10 + numeric_number) # sum numeric objects
print(intro + numeric_number) # this is not going to work
```

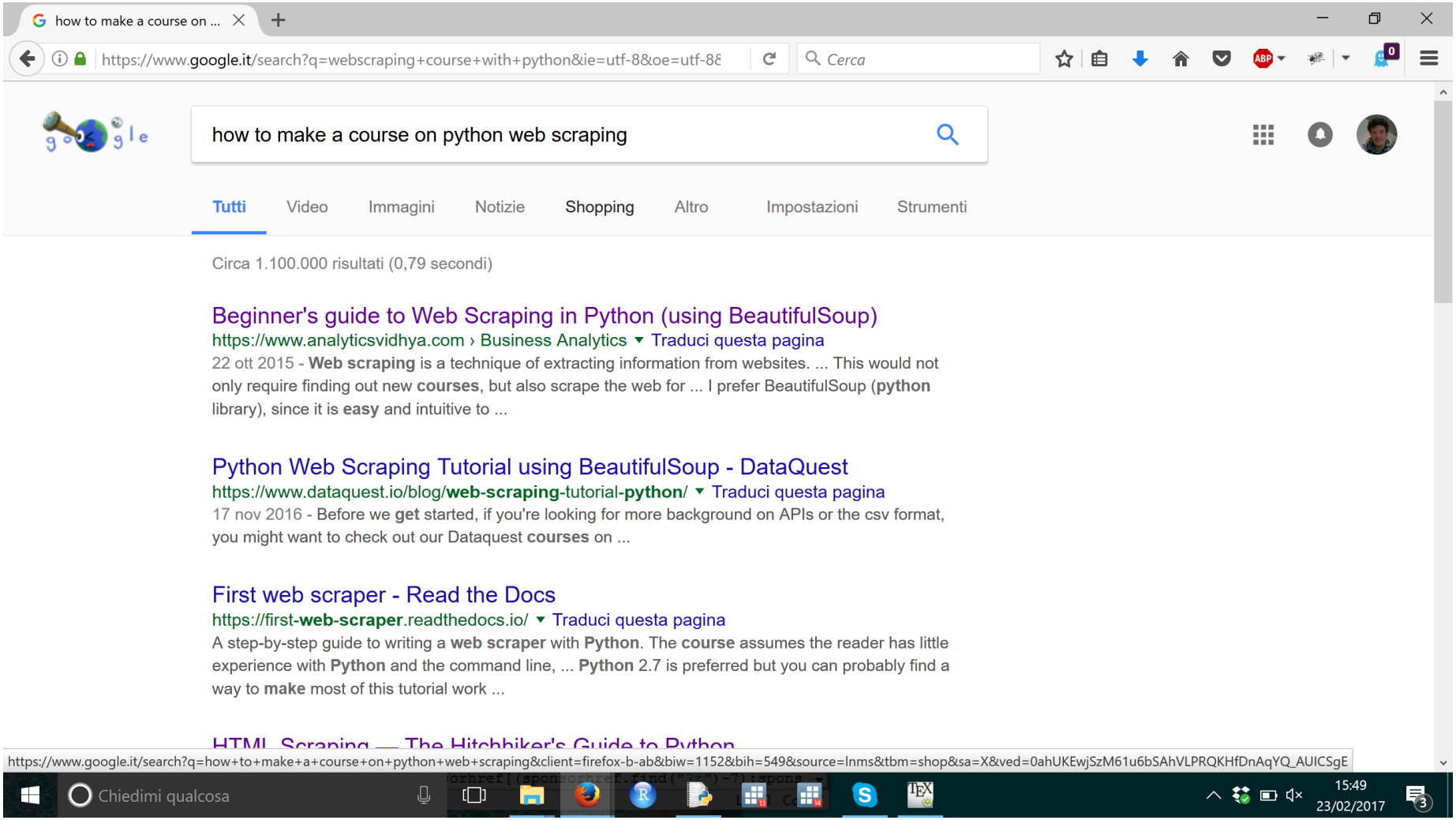
Python 2.7.11 Shell

File Edit Shell Debug Options Window Help

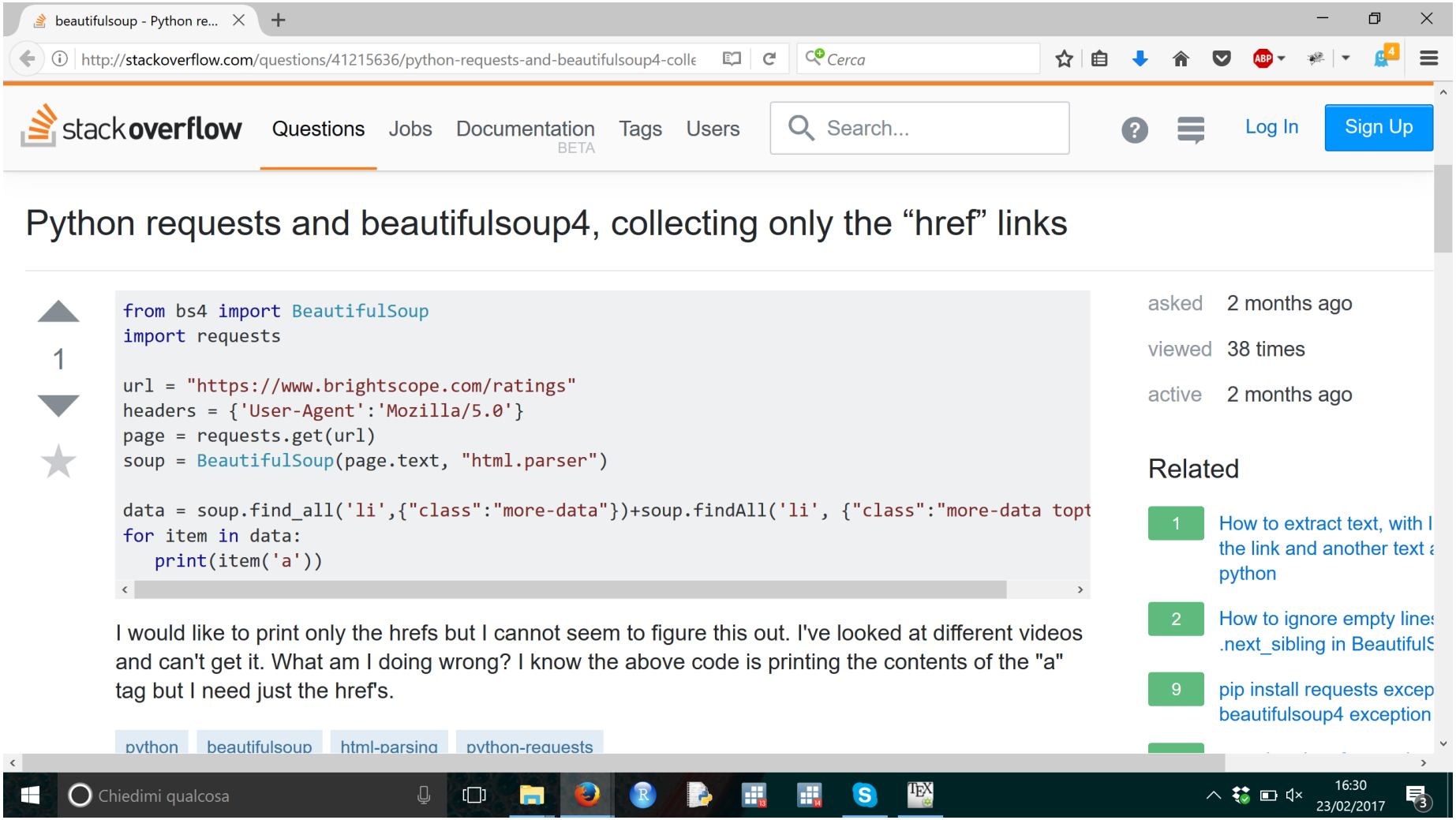
```
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec  5 2015, 20:32:19) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Users/gabriele/Desktop/foo.py ================
You can contatenate a string with a <type 'str'> object like 1
You can contatenate a string with a <type 'int'> object like 1 using str()
11

Traceback (most recent call last):
  File "C:/Users/gabriele/Desktop/foo.py", line 10, in <module>
    print intro + numeric_number # this is not going to work
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Best friends - Google



Best friends - Stack Overflow



Why Python?

- Python is a high-level, interpreted language
- No need to pre-specify the variable type
- "Poor" syntax - faster to learn
- Clear, readable, understandable
- Freely available, very popular (i.e., a lot of support!)
- Not good for memory-intensive tasks, not as fast as C/C++
- Continuously evolving, extremely flexible (from machine learning to webscraping)

Python Interactive Console

After installing Python Ubuntu Windows 10

- Linux/Mac: from terminal
- Windows: from prompt

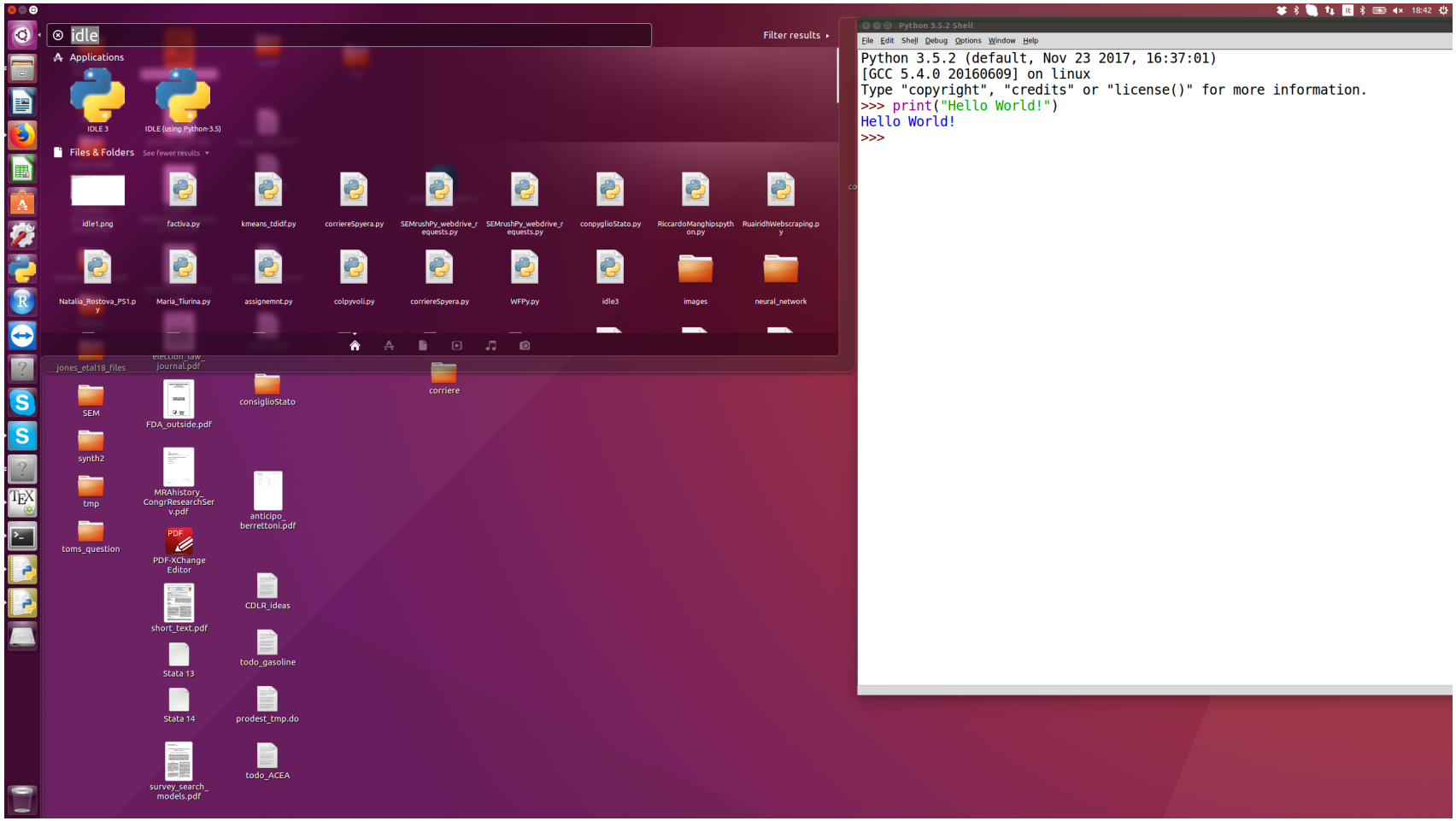
Typing $python in the terminal/prompt should yield

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more info
>>>
```

Different python versions (type $python2.7 or $python3)

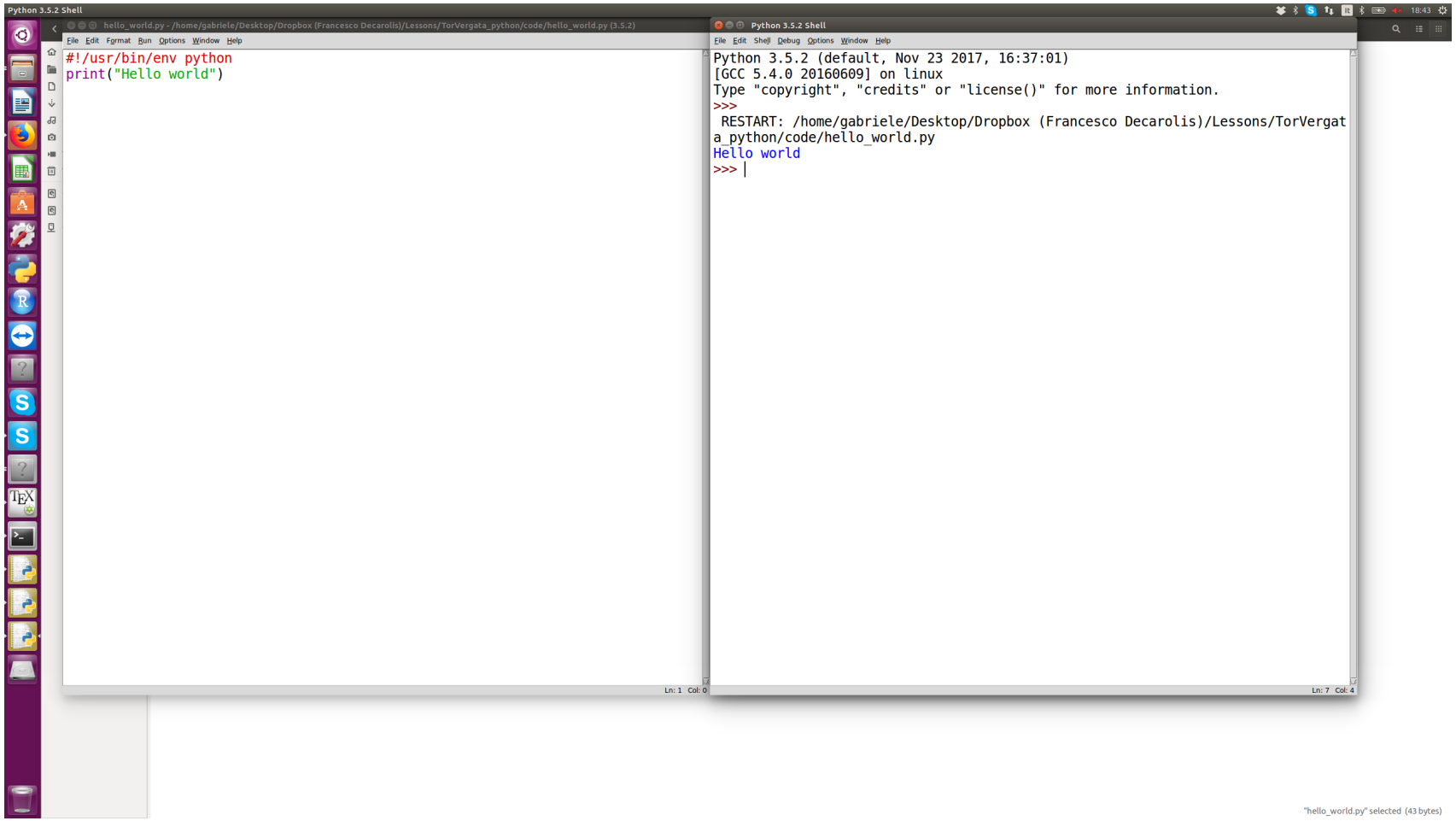
Python IDLE/Shell

The Integrated Development and Learning Environment comes by default with Python



Write Codes: Hello World!

```
#!/usr/bin/env python
print("Hello world")
```



Strings

```
1  # this is a string
2  str_sentence = 'This is a very simple string'
3  # this is a string, too, but multi-line
4  print('''How many lines must
5  a multi-line string have?''')
6  # this is a multi-line string with different quotes
7  print("""Let's see whether we can make
8  a multi-line object as well""")
9  # print() and concatenate methods
10 x = 'heaven'
11 z = 'hell'
12 print(x)
13 print("Can you tell %s from %s?" % (x,y))
```

Strings are **iterable** objects Iterables - Definition

Numbers

```
1  # Integers
2  birth_year = 1899
3  birth_year_from_string = int0("1899")
4
5  # Floating Points
6  pi = 3.14159265
7  pi = float("3.14159265")
8
9  # Fixed Points
10 from decimal import Decimal
11 price = Decimal("0.02")
```

Lists

The most basic data structure in Python is the sequence. Each element of a sequence is assigned a number - its position or index. The first index is zero, the second index is one, and so forth. The most common sequences are the lists.

```
1  # Initiate empty lists
2  empty_list = []
3  numeric_list = [1,1,2,3,5,8]
4  string_list = ['Fibonacci','Series','In','List']
5
6  # append() method
7  numeric_list.append(13)
8
9  # extend() method
10 numeric_list.extend([21,34])
```

append() vs. extend()

Working with Lists

```
1 numeric_list = [1,1,2,3,5]
2
3 # length of lists
4 len(numeric_list)
5
6 # identify list elements
7 numeric_list[0]
8 numeric_list[0:2]
9 numeric_list[4:5]
10
11 # multiple elements with different indices: for loop trick
12 indices = [0,3]
13 new_list = [ numeric_list[i] for i in indices ]
```

Dictionaries

Dictionaries are (unordered) collections of items. Dictionary elements are `key:value` pairs. `keys` can be integers or strings.

```
1          # dict() method
2 my_dict = dict()
3 # or curly brackets
4 my_dict = {
5 1 : 'third',
6 2 : 'second',
7 3 : 'first'}
8 # keys can be integers or strings
9 tcid_ym = {'first' : 3, 'second': 2, 'third' : 1}
10
11 print(my_dict[1])
12 print(tcid_ym['first'])
```

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.

```
1 for iterating_var in sequence:
2 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

```
1 for letter in 'Python':      # First Example
2    print('Current Letter :', letter)
3
4 fruits = ['banana', 'apple',  'mango']
5 for fruit in fruits:         # Second Example
6    print('Current fruit :', fruit)
7
8 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.

```
1 if expression:
2    statement(s)
3 else:
4    statement(s)
```

Dictionary Comprehension

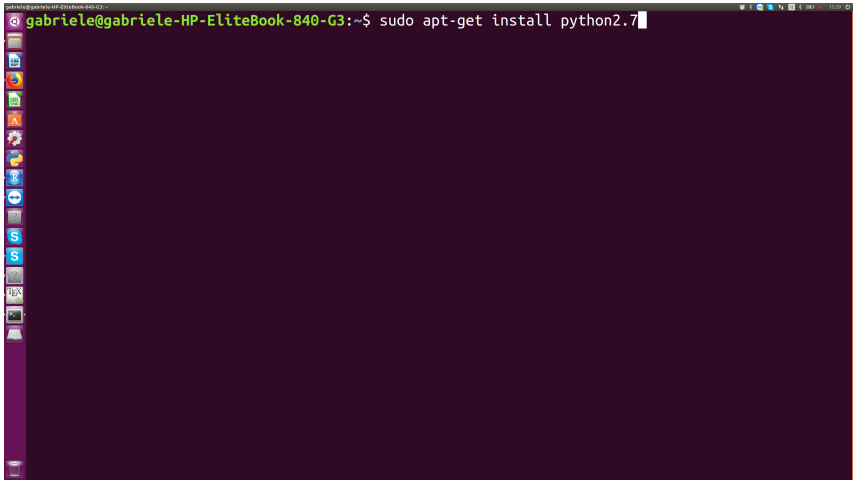
- is a concise way to generate a dictionary from an iterable object;
- consists of a (key: value) pair followed by *for* statement

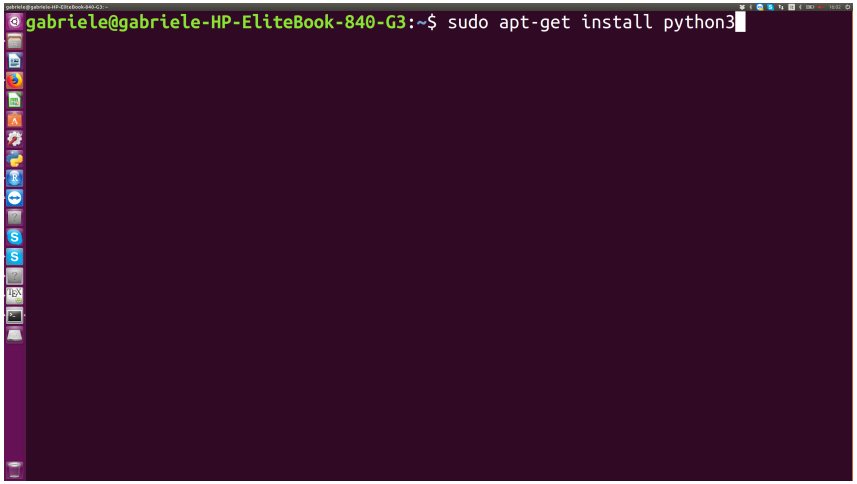
```
1  # generate the square of the fibonacci series
2  fibonacci_list = [1,1,2,3,5,8,13] # ITERABLE object
3  fibonacci_sq_dict = {x : x*x for x in fibonacci_list}
4  fibonacci_sq_dict # {1: 1, 2: 4, 3: 9, 5: 25, 8: 64, 13: 169} *** look at t
5  # the equivalent for loop reads
6  fibonacci_sq_dict_long = {} # initiate the dict
7  for x in fibonacci_list:
8  fibonacci_sq_dict_long[x] = x*x
9  # we may want to add more if and for statements
10 fibonacci_sq_dict_odd = {x : x*x for x in fibonacci_list if x%2 == 1}
11 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 create a dictionary from a string. Sample string : "python for webscraping" solution

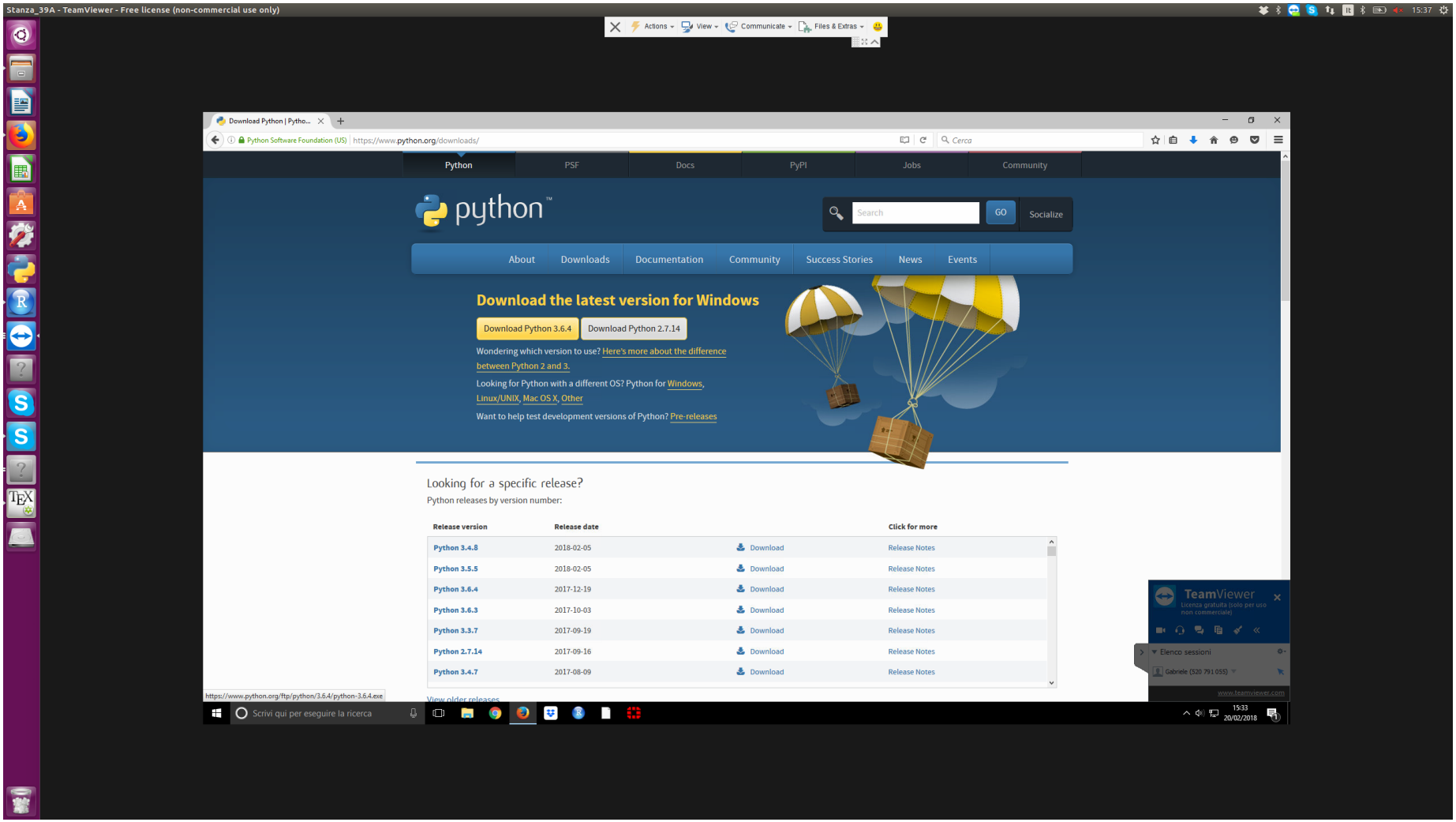
Ubuntu - Install





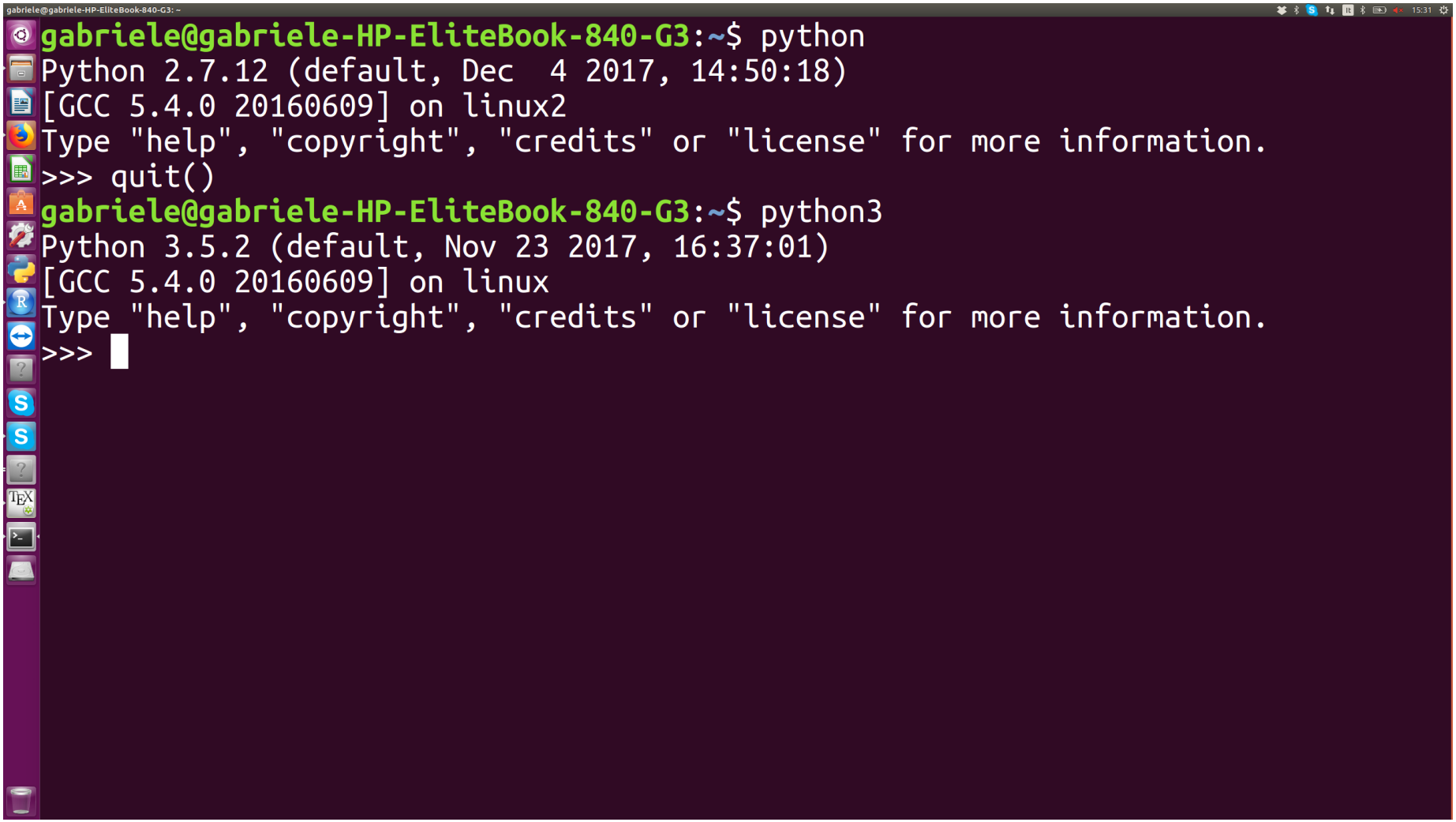
back

Windows - Install



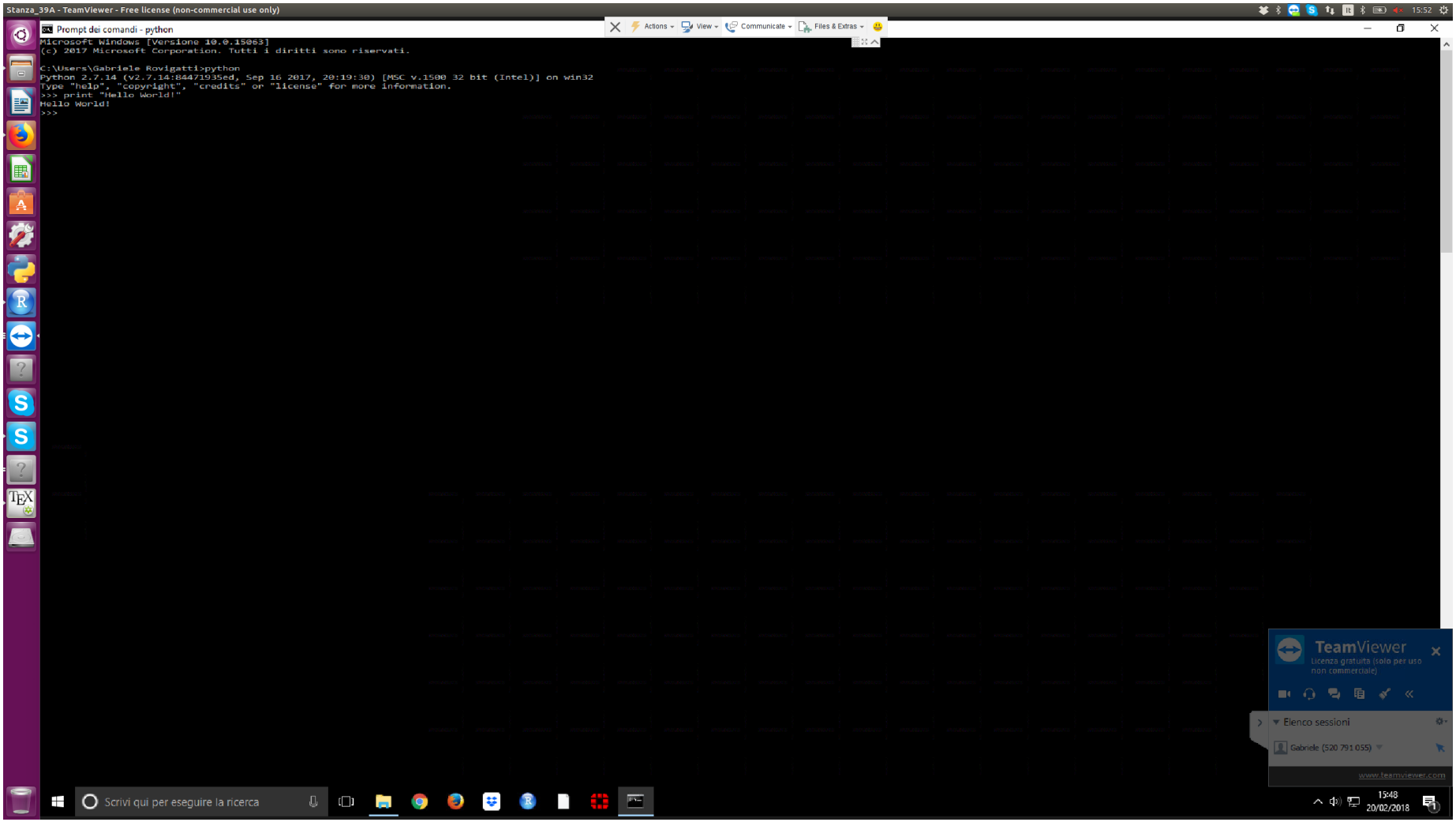
back

Linux - Terminal



back

Windows - Prompt



back

Definition of Iterable

Back

Iteration is a general term for taking each item of something, one after another. In Python, **iterator** and **iterable** have specific meanings.

- An **iterator** is an object with a `next` (Python 2) or `__next__` (Python 3) method:
  - Whenever you use a `for` loop, or `map`, or a list comprehension, etc. in Python, the `__next__` method is called automatically to get each item from the iterator
- An **iterable** is an object that has an `__iter__` method which returns an iterator, or which defines a `__getitem__` method that can take sequential indexes starting from zero (and raises an IndexError when the indexes are no longer valid)

append() vs. extend() method

The `list.append()` method appends **one** object to a list. The `list.extend()` method extends a list by appending elements from an iterable.

```
1 my_list = ['this','is']
2 addendum1 = ['a','list']
3 addendum2 = ['of','strings']
4 print(my_list.extend(addendum1))
5 print(my_list.append(addendum2))
```

back