SIRE504
Introduction to Python3

11.10.2018

Harald Grove

harald.gro@mahidol.ac.th

Outline

- Where and how to use Python

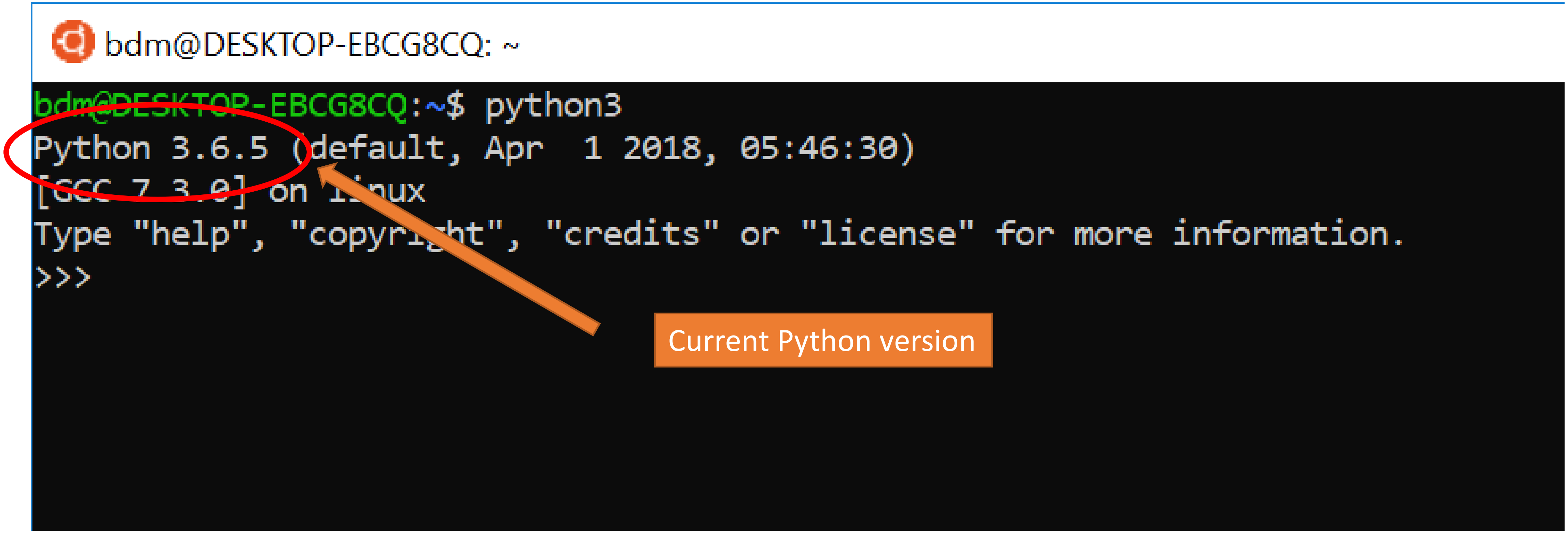
- Small example scripts

- Self study of Python syntax

Python console

- Start python3
  - $ python3

- Quit:
  - Ctrl+d or exit()
- Getting help (exit help: ‘q’):
  - help(print)

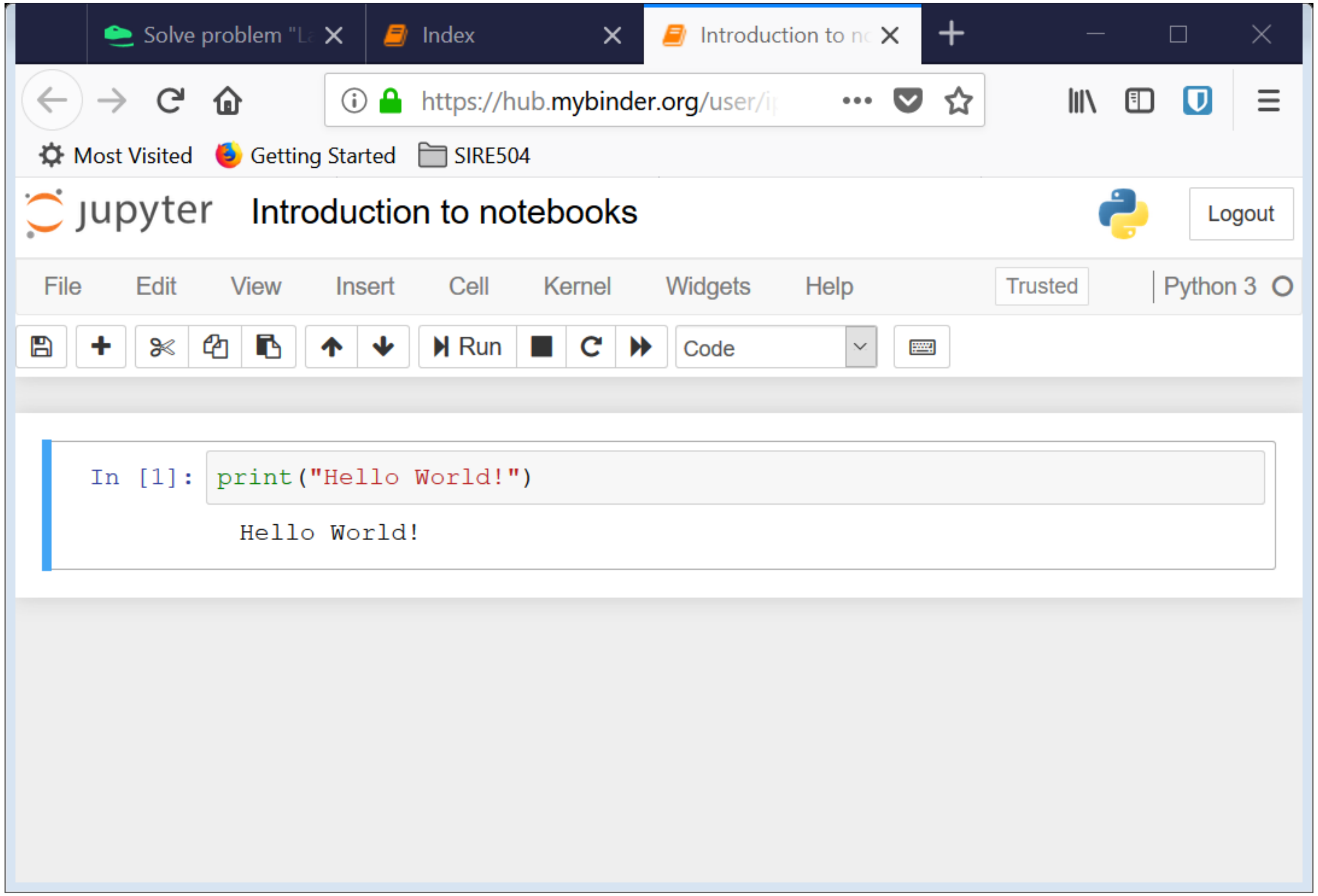


```
bdm@DESKTOP-EBCG8CQ:~$ python3
Python 3.6.5 (default, Apr  1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Current Python version

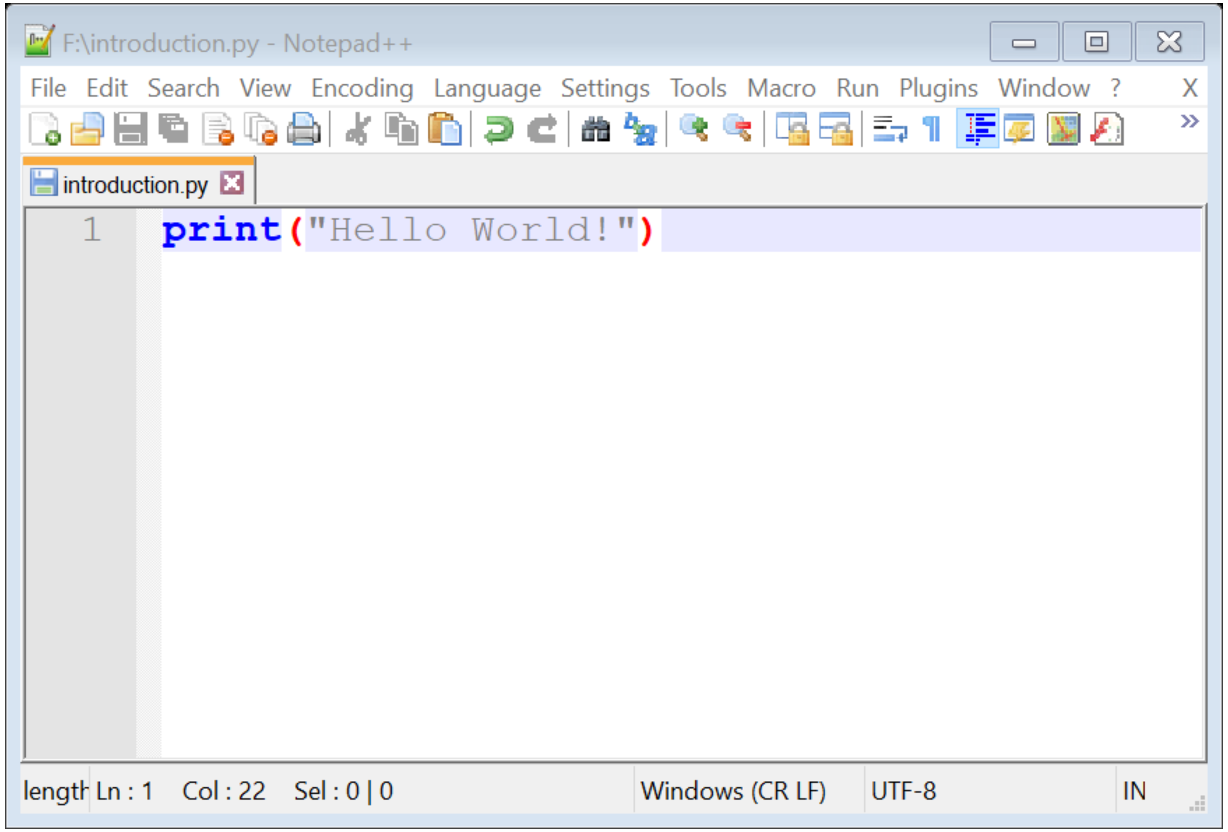
Jupyter notebook

- Try it out:
  - http://jupyter.org/try
- Easy install
  - Anaconda
  - https://www.anaconda.com/download/



```
In [1]: print("Hello World!")

        Hello World!
```

A text editor and the command line

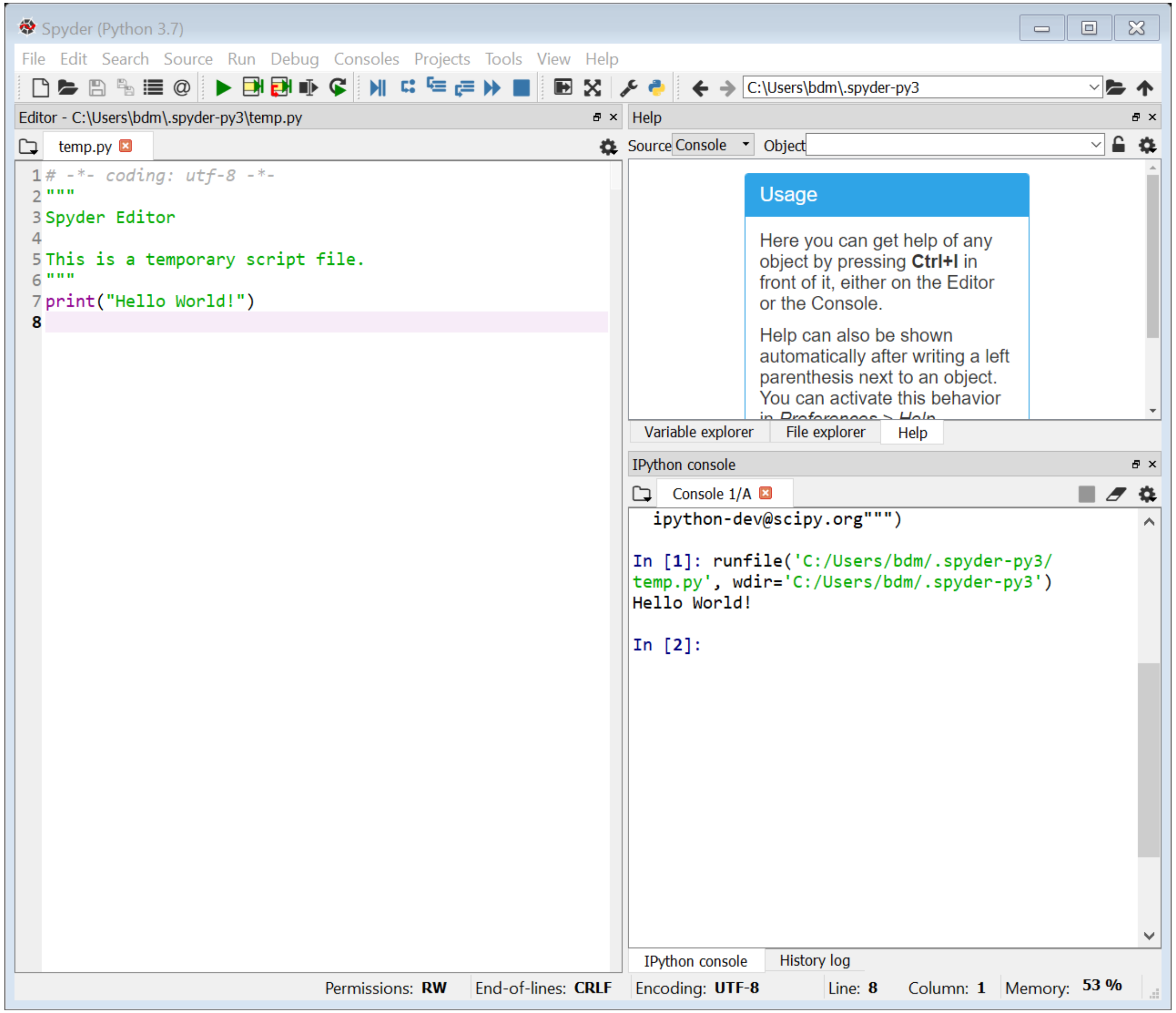

F:\introduction.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? X
introduction.py
length Ln : 1 Col : 22 Sel : 0 | 0
Windows (CR LF)
UTF-8
IN

```python
print("Hello World!")
```

- Python aware editors:
  - Files saved as “.py” are displayed with syntax highlighting.
- Run the program:
  - $ python3 introduction.py
- Options:
  - Vim, Nano, Emacs, Notepad++

Integrated development environment (IDE)

- Syntax highlighting
- Error checker/highlighter
- Integrated console
- Framework for handling projects
- Options:
  - PyCharm, PyDev (Eclipse), Wing IDE, Komodo IDE, Eric, Spyder, PyScripter, Python-IDLE.



```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
print("Hello World!")

```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

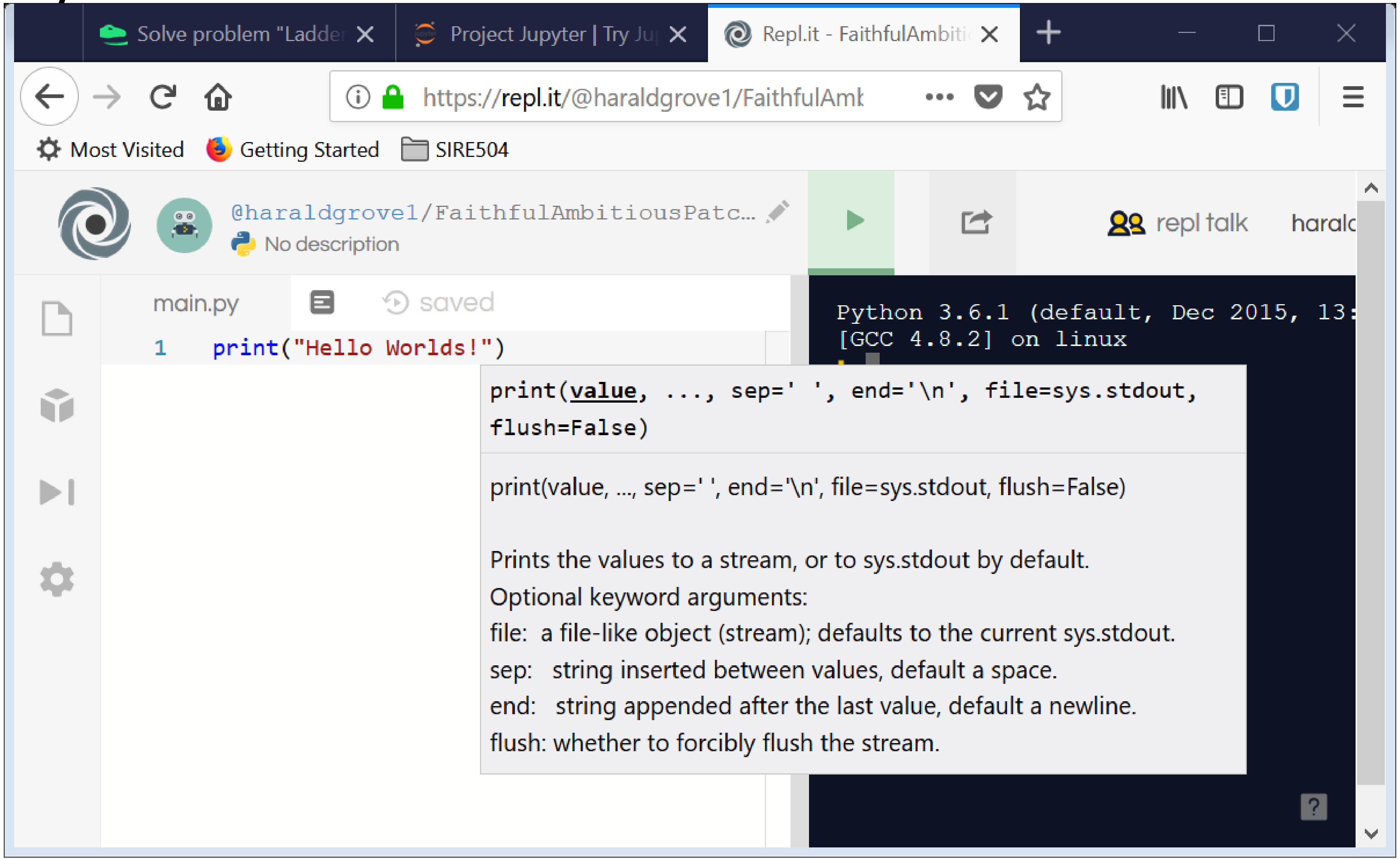
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

```
 ipython-dev@scipy.org""")

In [1]: runfile('C:/Users/bdm/.spyder-py3/
temp.py', wdir='C:/Users/bdm/.spyder-py3')
Hello World!

In [2]:
```

Permissions: **RW** End-of-lines: **CRLF** Encoding: **UTF-8** Line: **8** Column: **1** Memory: **53 %**

Online Python editors

- repl.it



```
1   print("Hello Worlds!")
```

```
Python 3.6.1 (default, Dec 2015, 13:
[GCC 4.8.2] on linux
```

```
print(value, ..., sep=' ', end='\n', file=sys.stdout,
flush=False)
```

print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.

The first program (expanded)

```
greeting = “Hello World!”
print(greeting)
```

Variable

- Variables are a way to assign names to objects
  - Text, numbers, collections
  - Methods
- Useful when you want to reuse an objects several times.

Naming variables

- Legal names
  - Starts with a letter or ‘_’
  - The rest of the name can be letters, numbers or ‘_’
- Recommendation
  - Try to make the name descriptive, but don’t overdo it.
  - ‘name’, ‘height’, ‘weight’ is easier to understand than ‘a’, ‘b’, ‘c’.
  - All lowercase letters is a widely used standard.
  - Avoid using ‘_’ at beginning or end of variable names.

A peek behind the scenes

```
    greeting = “Hello world!”
    print(greeting)
```

- What is “greeting” here?

```
    type(greeting)
    dir(greeting)
```

- **type** tells you what the *class* of the object is
- **dir** tells you what the *attributes* and *methods* of the class are

Classes

- A class is an *object* that:
  - stores some data
  - can do something with this data (methods)
  - is created using a constructor, e.g. str()

Like a bread-baking machine:

- Add ingredients (data)
  - Flour
  - Water
  - Yeast
- Make the bread (Methods):
  - Mix ingredients
  - Raise the dough
  - Bake the dough.

Creating an “*instance*” of a class

- Most classes are created by using the constructor, e.g. str()
- Some basic classes are a special case:
- This:
  - greeting = “Hello World!”
- is the same as this:
  - greeting = str(“Hello World!”)

This is the standard way of creating a new instance.

Classes can have many methods, use them

- Exploring the string class (str)
  - 77 methods (`>>> len(dir(greeting)`)
  - Use help() to see the documentation (`>>> help(str.split)`)
- Some things a string class can do:
  - Format itself to upper case (`>>> greeting.upper()`)
  - Split by words (`>>> greeting.split()`)
  - Count the occurrences of a character (`>>> greeting.count(“o”)`)
  - Center itself in a width (`>>> greeting.center(30)`)
  - … and many more

Different types of methods

Special methods:

[‘_ _add_ _’,

‘_ _class_ _’,

‘_ _contains_ _’, …

Public methods:

‘rstrip’,

‘split’,

‘splitlines’, …]

- Special methods are very rarely used directly.
- Focus on public methods.

Class methods can return new classes

```
>>> greeting = “Hello World!”
>>> words = greeting.split()
```

Which class does “words” belong to?

```
>>> type(words)
>>> dir(words)
```

Other classes

Data content	Name	Constructor	Methods	Public methods
Text	String	str()	77	44
Whole numbers	Integer	int()	70	8
Floating point numbers	Float	float()	57	7
Linear collections	List	list()	46	11
Indexed collections	Dictionary	dict()	40	11
True/False	Boolean	bool()	70	8

Programming = Problemsolving

- Procedureal task: learn the sequence of activities that allow you to achieve a goal.
- Conceptual task: understand the principles that govern the domain and interrelations between pieces of knowledge.

Example problem

- Find the two largest elements of a list of positive integers

- With this input:
  - 1, 3, 1000, 2
- The program should return:
  - 1000, 3

A procedural solution

1. Set max1, max2 to 0
2. If there are no remaining elements, stop.
3. Take the next element, call it value.
4. Find the smallest of max1 and max2 and if it is smaller than value set it to value.
5. Go to step 2

A procedural solution in Python

```
max1 = max2 = 0
values = [1, 3, 1000, 2]
for value in values:
        if max1 < value < max2:
                max1 = value
                continue
        if max2 < value < max1:
                max2 = value
                continue
        if max1 < value and max1 <= max2:
                max1 = value
                continue
        if max2 < value and max2 <= max1:
                max2 = value
                continue
print(max1, max2)
```

A conceptual solution

1. Sort the list in descending order
2. Take the first two elements

```
values = [1, 3, 1000, 2]
values = sorted(values, reverse=True)
max1, max2 = values[:2]
print(max1, max2)
```

Data structures are important

- Which data structure solves my problem right away?

- Every solution means applying the right concepts and data representation to the problem.

Lab work / homework

- Review chapters 1 – 7 on snakify.org
  1. Input, print and numbers
  2. Integer and float numbers
  3. Conditions: if, then, else
  4. For loop with range
  5. Strings
  6. While loop (optional)
  7. Lists
- rosalind.info
  - Python problems
  - Bioinformatics problems using python

Python syntax

A more detailed look at the syntax of Python functions and methods

Python basics

- Remember indentation!
- Use spaces, not tabs.
    - Tabs are not guaranteed to display the same in all editors.

Input, print and numbers

- Assignment
  - variable = value
- Print content to screen:
  - print(s1[,s2,…,sn] [, end=<end-of-line-char>] [, sep=<between-word-char>])
- Data types:
  - Whole numbers, decimal numbers, text: int() , float(), string()
- Basic operators:
  - Addition (+), subtraction (-), multiplication (*), division (/)

Conditions: if, then, else

- Syntax for making a choice:
  - if <condition1, that can be either True or False>:
    - <Statement if condition1 is True>
  - elif <condition2, that can be either True or False>:
    - <Statement if condition2 is True>
  - else:
    - <Statement if none of the conditions are True>
- Comparison operators
  - less (<), greater (>), less or equal (<=), greater or equal (>=), equal (==), not equal (!=)
- Logical operators
  - Both (and), Either (or), Negate (not)

Integer and float numbers

- Convert from float to integer
  - Discard fraction (int(f)) or round towards zero (round(f))
- Advanced operators
  - Exponention (**), Integer division (//), Modulo (%)
- Scientific notation
  - 1.93e11, 1.93e-11
- Precision of floating number
  - 0.1 + 0.2 != 0.3
  - Do not use equal (==) or not equal (!=) with floating point numbers.
- Math module
  - import math
  - math.ceil(f), math.floor(f), math.sqrt(f), math.log(f), math.pi, math.e

For loop with range

- Repeat an action multiple times
    - for <variable> in <sequence>:
        - <Statement>
    - else:
        - <Statement if loop finished normally>
- Sequence can be anything that contains distinct elements:
    - strings, lists, dictionaries, files
- Looping a certain number of times
    - range([start,] end [,step])
    - One number: end (start defaults to 0 and step to 1)
    - Two numbers: start, end (step defaults to 1)
    - Three numbers: start, end, step
    - Returns numbers in range [start, end>

Strings

- Sequence of characters
  - Length (len(s))
- Indexing (position in sequence)
  - First letter has index 0, last has index -1

s[0]	**s[1]**	**s[2]**	**s[3]**	**s[4]**
H	e	l	l	o
s[-5]	s[-4]	s[-3]	s[-2]	s[-1]

- Slice (subset of the sequence)
  - s[start:stop:step]
  - Similar to range, one number is stop, two numbers are start and stop.

Strings, part2

- Strings are immutable
  - All edits should be assigned to a new variable
- String methods
  - Locate a substring in a longer string, s.find(substring), s.rfind(substring)
  - Replace a substring with another string, s.replace(substring1, substring2)
  - Count the number of occurences of a substring, s.count(substring)
- Special characters
  - tabulator: \t
  - line shift: \n

Strings, part3

- Adding variables into a string:
  - '{} {} {}'.format(var1, var2, …, varn)
  - Variables are matched to {} based on position.
- Example code:
  - num_apples = 10
  - weight_sugar = 1
  - print("I need {} apples and {} kg of sugar".format(num_apples, weigth_sugar))
- Output:
  - I need 10 apples and 1 kg of sugar

While loop

- Repeating an action an unknown number of times
  - while <condition that is either True or False>:
    - <Statement if condition is True>
  - else:
    - <Statement if the loop finishes normally>
- Loop controls
  - Exit the loop prematurely, skipping any remaining statements (break)
  - Skip the remaining statements, and move to the next iteration (continue)

Lists

- A linear collection of objects
  - Syntax: [v1, v2, …, vn]
- List of the same character
  - [0]*n = [0, 0, 0, …. , 0]
- Convert to and from string (Excel: text-to-columns):
  - s.split(<seperator character>, <number of splits to perform>)
  - <string to insert between the elements in the list>.join(<list>)
- Generators
  - [<statement> **for** <variable> **in** <sequence>]

Lists, part 2

- Lists works like strings for indexing and slicing.
  - L[index], L[start:end:step]
- List operations
  - If an element is in the list, x in L
  - If an element is not in the list, x not in L
  - The smalles element in the list, min(L)
  - The largest element in the list, max(L)
  - The position of element in the list, A.index(x)
  - The number of occurrences of element in the list, A.count(x)

Lists, part 3

- Loop over lists, getting both the index and the value (enumerate):
  - for index, value in **enumerate**(list):
    - Statement
- Lists are mutable, values in a list can change "in-place".
- Functions working on lists can either:
  - return a new list with the changes: e.g. sorted(list)
  - change the list "in-place": e.g. list.sort()
  - Example, sorting a list:
    - In-place: list.sort()
    - Return new list: sorted(list)