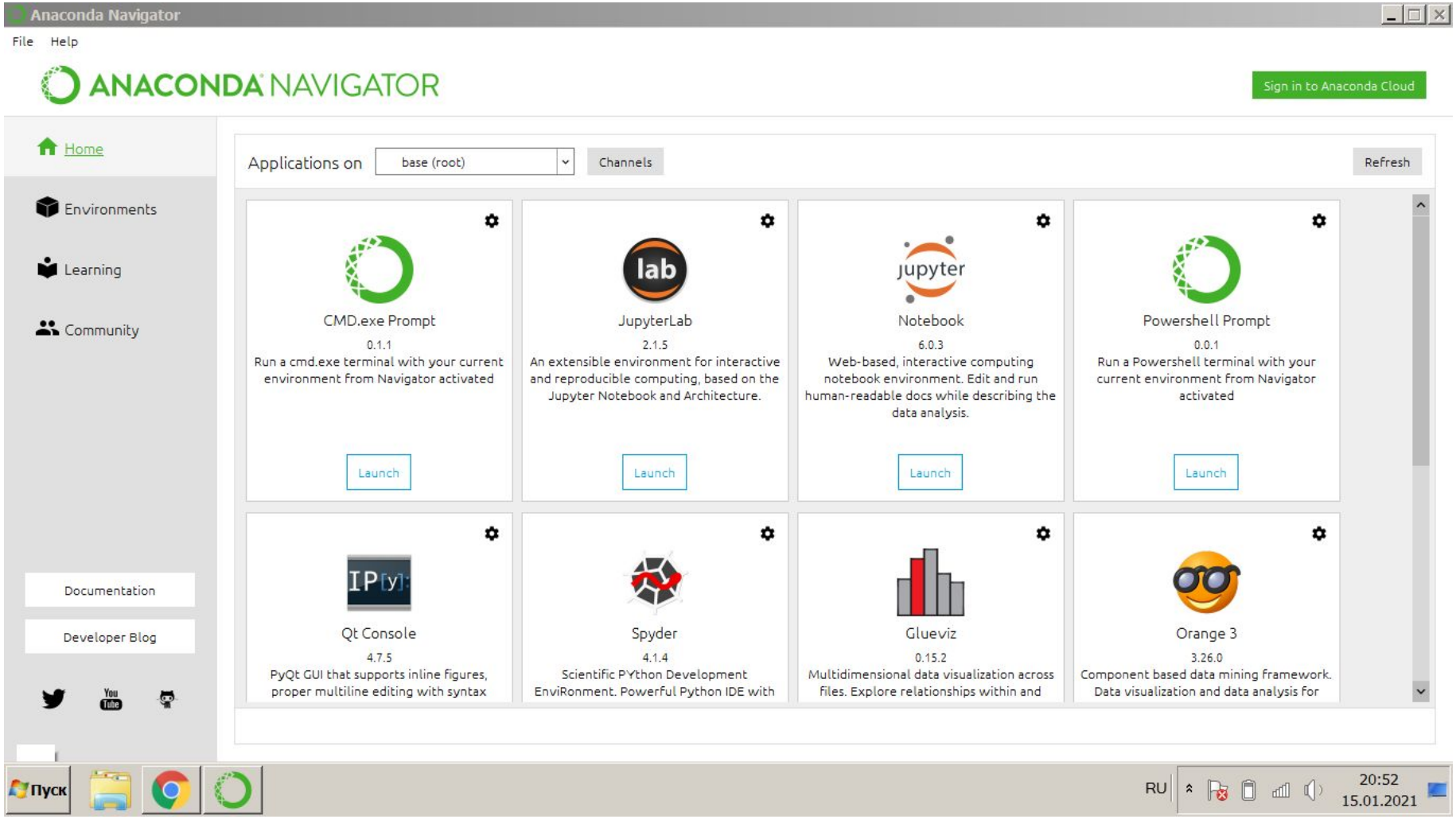
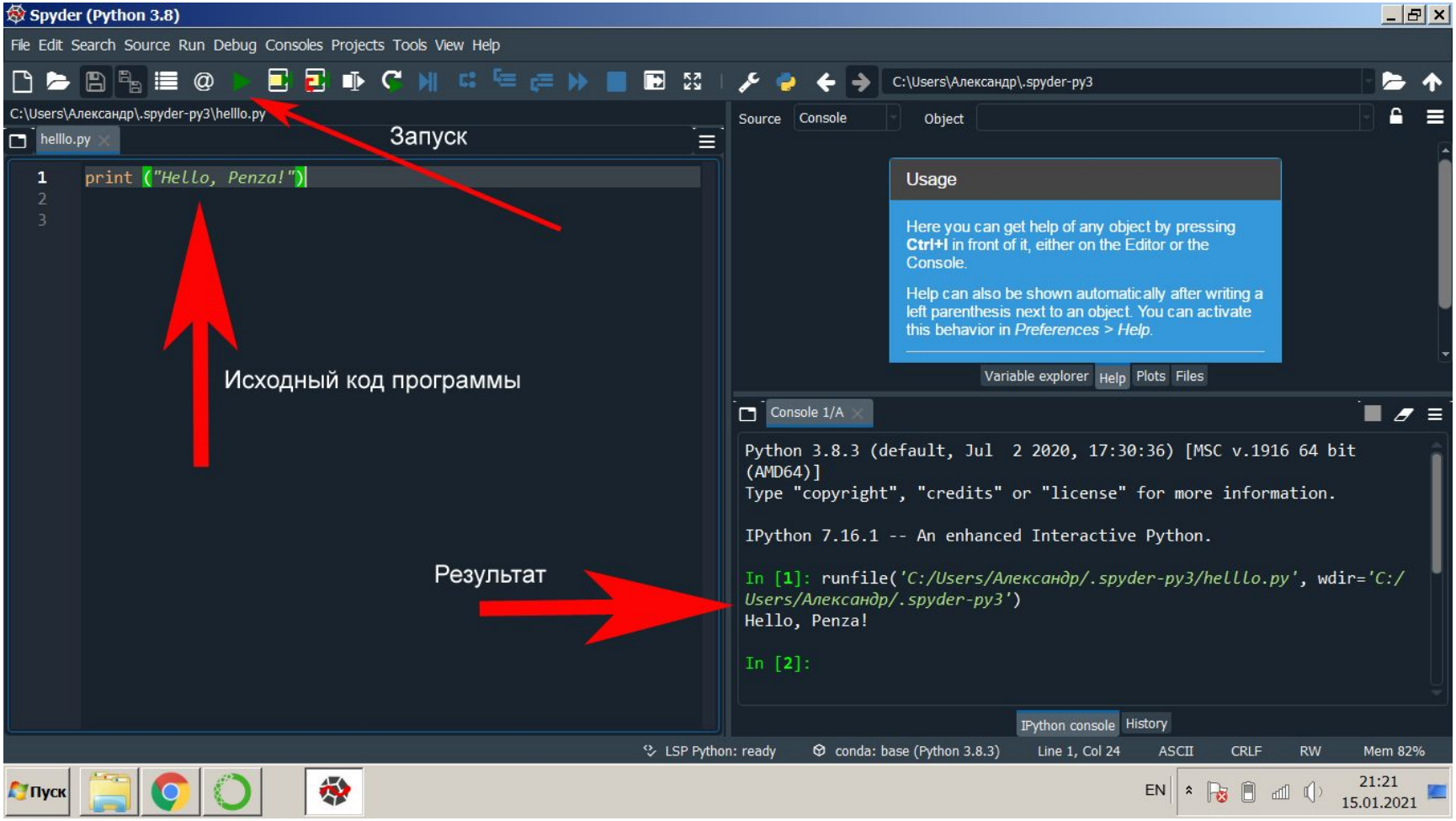
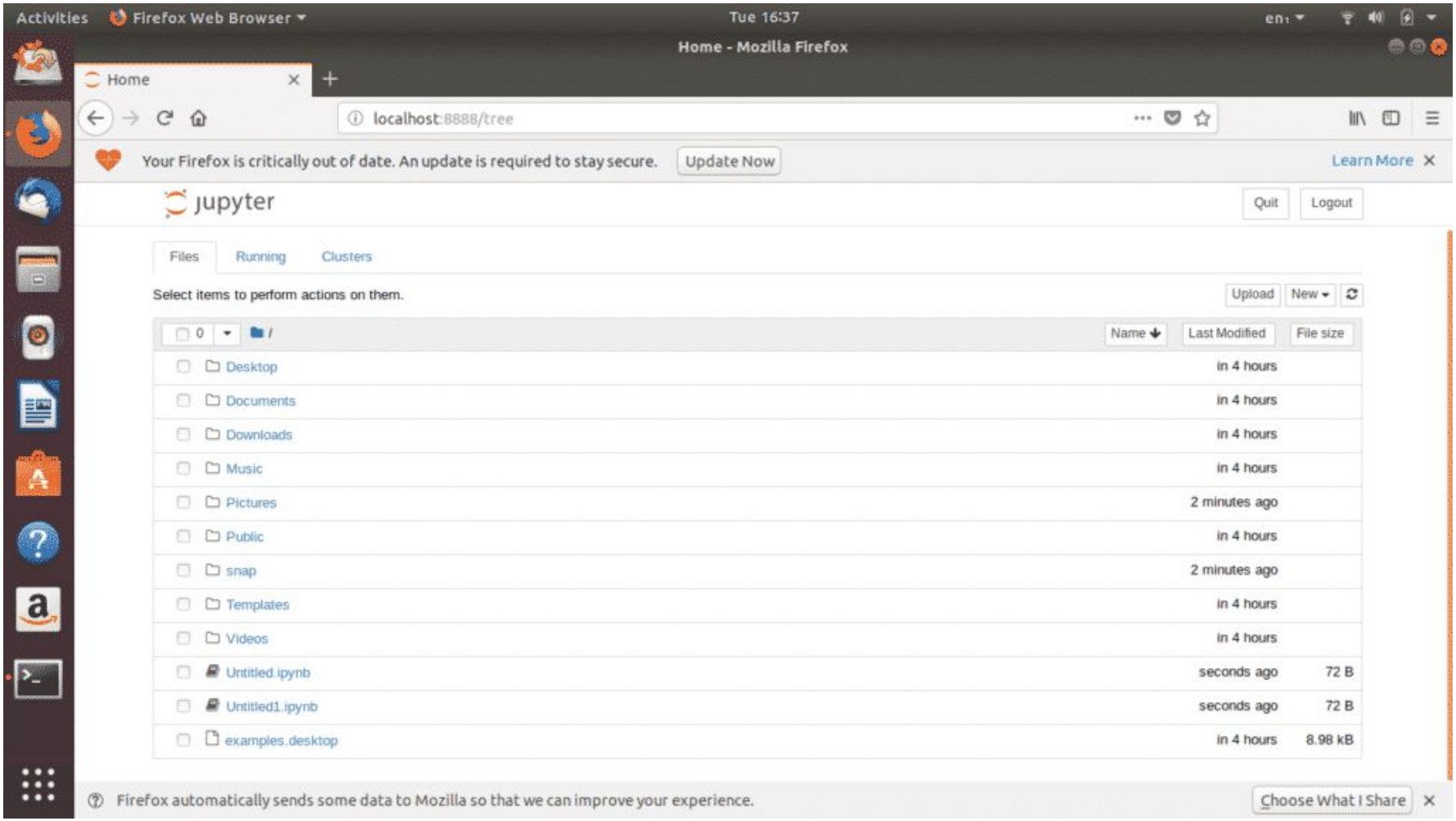

Anaconda Navigator
File Help
ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Learning
Community
Documentation
Developer Blog
Applications on
base (root)
Channels
Refresh
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
JupyterLab
2.1.5
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.0.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
Qt Console
4.7.5
PyQt GUI that supports inline figures, proper multiline editing with syntax
Spyder
4.1.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with
Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and
Orange 3
3.26.0
Component based data mining framework. Data visualization and data analysis for
Пуск
RU
20:52
15.01.2021

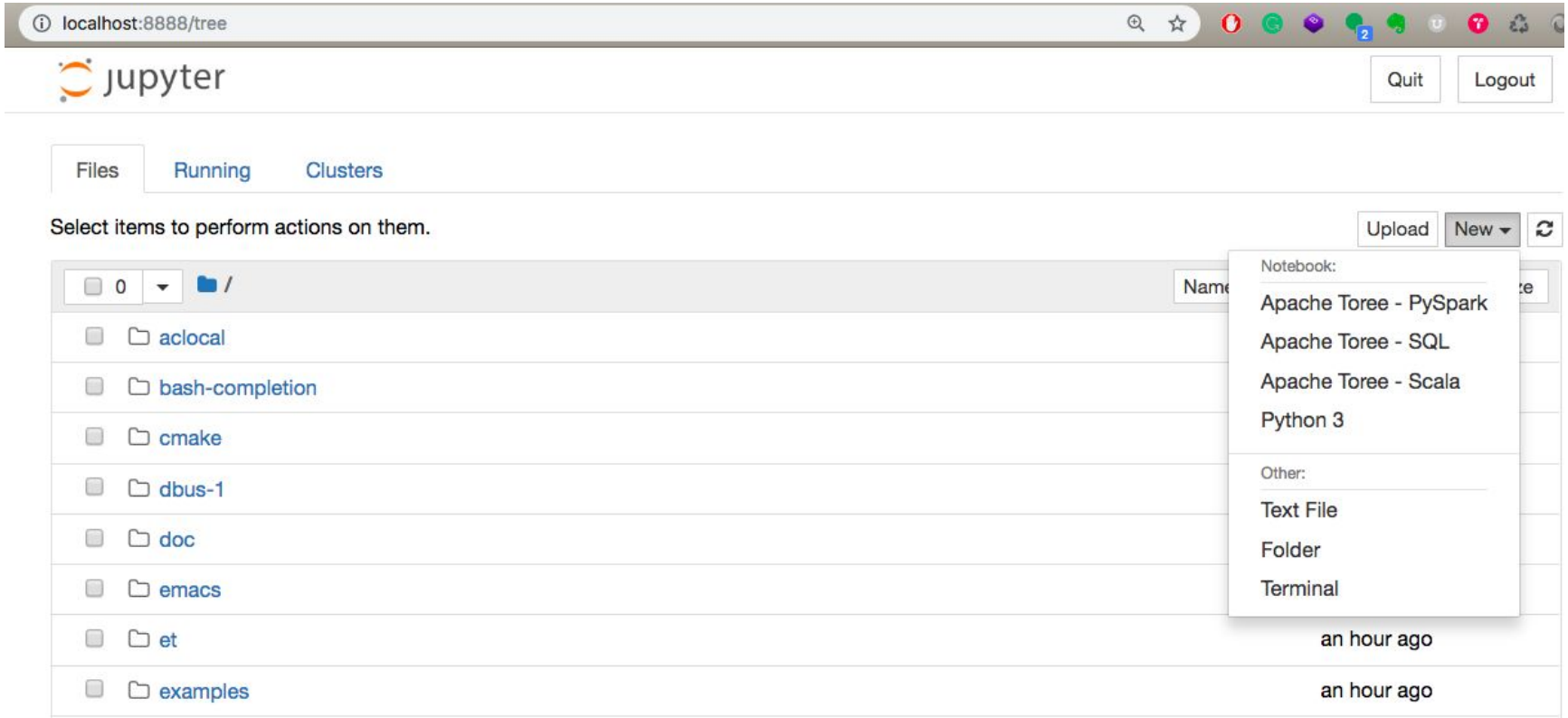
Spyder


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Александр\.spyder-py3
C:\Users\Александр\.spyder-py3\helllo.py
helllo.py
Запуск
1 print ("Hello, Penza!")
2
3
Исходный код программы
Результат
Source Console Object
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.
Variable explorer Help Plots Files
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/Александр/.spyder-py3/helllo.py', wdir='C:/Users/Александр/.spyder-py3')
Hello, Penza!
In [2]:
IPython console History
LSP Python: ready
conda: base (Python 3.8.3)
Line 1, Col 24
ASCII
CRLF
RW
Mem 82%
Пуск
EN
21:21
15.01.2021

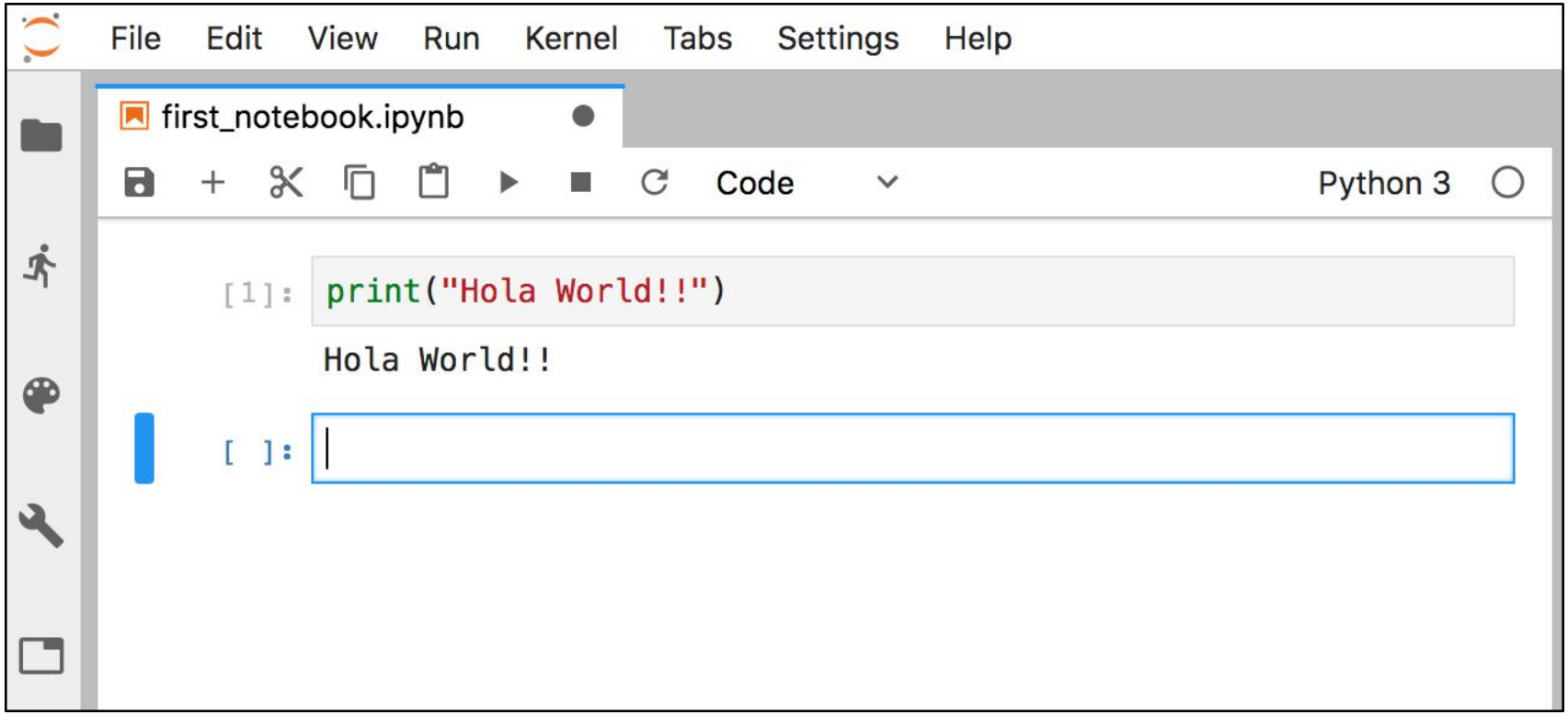
Jupyter Notebook (общий вид)



Jupyter Notebook
(создание нового проекта)



Jupyter Notebook
(работа с notebook)



Быстрое введение в синтаксис языка Python

Арифметические операции

- Сложение: +
- Вычитание –
- Умножить: *
- Делить: /
- Делить нацело: // (например: 10//3=3)
- Остаток от деления: % (например: 10%3=1)
- Возведение в степень: ** (например, 10**3=1000)
- Модуль числа x: abs(x)

Быстрое введение в синтаксис языка Python

Типы данных

- Наиболее распространенными типами данных в Python являются числа и строки.
- *Числа* различают:

целые int (например: 1, 100, -5);

дробные float (например: 3.2, -3.2).

комплексные complex.

- *Строки*, это любые символы заключенные в кавычки (тип str). Допускаются как двойные, так и одинарные кавычки. Например: “Hello”, ‘Hello’.
- Существуют также другие встроенные типы (булевый, бинарные, последовательности и т.п.), информацию о которых можно найти в любом справочнике по Python.

Быстрое введение в синтаксис языка Python

Строковые операции

‘a’+’bb’ = ‘abb’

Комментарии

Комментарий в коде программы на Python задаются с использование символа #. Например:

```
# Это комментарий
x=5 # И здесь может быть комментарий
```

Переменные

- В языке Python не требуется специального объявления переменных и указания их типа. Переменная создается в момент первого использования. Тип определяется из контекста этого использования.

Например:

```
x=5 # Создается числовая переменная x, в которую записывается   число 5
y=”Hello” #Создается строковая переменная y со строкой “Hello”
```

- Если нужно явно указать тип переменной, то может быть использован механизм кастинга (casting). Например,

```
y=str(5) # строковая переменная y со строкой “Hello”
x=int(5) # создается числовая переменная x, в которую записывается число 5
z=float(5) # создается числовая дробная переменная y, в которую записывается 5.0
```

- В любой момент можно получить тип переменной с помощью функции type. Например:

```
y=str(5)
print (type(y)) # вывод на экран тип  переменной y
```

- Имена переменных чувствительны к регистру! Например,

```
x=5
X=”Hello”
# x и X – это разные переменные
```

Операции сравнения

С помощью операции сравнения можно сравнить два числа, получив результат True или False. Например:

- Больше (3>2 #Результат True);
- Меньше (3<2 #Результат False);
- Больше или равно (3>=2 #Результат True);
- Меньше или равно (3<=2 #Результат False);
- Равно (3==2 #Результат False);
- Не равно (3!=2 #Результат True).

Как правило, операции сравнения используются в условных конструкциях.

Операторы условия (условные конструкции)

- Общий вид простого условия в Python следующий:

```
if <условие>:
    <действия, выполняемые в случае истинности условия>
```

Например:

```
if x>y:
    print (“x больше, чем y”)
```

Двоеточие и перевод строки для действий – это обязательные условия синтаксиса!
Нарушение этого правила приведет к синтаксическим ошибкам.

- Условия могут быть более сложными – с двумя и более ветвями:

```
if <условие>:
    <действия, выполняемые в случае истинности условия>
else:
    <действия, выполняемые в противном случае>
```

Например:

```
if x>y:
    print (“x больше, чем y”)
else:
    print (“x меньше или равно, чем y”)
```

Операторы условия (условные конструкции)

- Условие с тремя ветвями

```
if <условие>:
    <действия, выполняемые в случае истинности условия>
elif:
    <действия, выполняемые, если первое условие не выполнено>
else:
    <действия, выполняемые, если все предыдущие условия не выполнены>
```

Например:

```
if x>y:
   print (“x больше, чем y”)
elif x<y:
   print (“x меньше, чем y”)
else:
   print (“x равно y”)
```

Обратите внимание, для использования else обязательно должна быть ветка if или elif выше!

Логические операции

- Логические операции работают с логическими значениями True и False. Используются, как правила, для построения сложных логических выражений в условных операторах. В Python используется три основных логических операции:

not – инверсия;

and – логическое умножение “И” (возвращает False, если хотя бы один из операндов False);

or – логическое сложение “ИЛИ” (возвращает True, если хотя бы один из операндов True).

- Логические действия можно соединять друг с другом и указывать порядок выполнения операций с помощью скобок. Например:

True and (False or True) # Результат True

Списки (массивы)

Список это сложный тип данных, позволяющий хранить несколько значений одного или разных типов. Для его создания необходимо перечислить внутри квадратных скобок [] значения через запятую. Например:

Names = [‘Ivan’, Maria’, ‘Alex’]

Man = [‘Ivan’, ‘Petrov’, 33]

- Список можно изменять, добавляя новые значения с помощью метода .append(). Например:

  Names.append(‘Petr’) #Результат [‘Ivan’, ‘Maria’, ‘Alex’, ‘Petr’]

- Обращение к элементам списка происходит с помощью индексов в квадратных скобках. Индексация списка начинается с 0 (индекс первого элемента списка 0). Например:

  Names [0] # Результат ‘Ivan’

  Names [1] # Результат ‘Maria’

- Возможна индексация с обратного конца списка! Для этого используются отрицательные индексы. Например:

  Names [-1] # Результат ‘Petr’

Списки (массивы)

- Возможно получить доступ сразу к нескольким элементам списка (так называемые, срезы или слайсы списка). Общий синтаксис среза такой:

[start:stop:step]

где, **start** – от какого элемента включительно(по умолчанию 0);

stop – до какого элемента включительно (по умолчанию – последний элемент списка);

step – с каким шагом (по умолчанию - 1).

Например:

```
# От элемента с индексом 1 включительно до конца списка
Names [1:] # Результат [‘Maria’, ‘Alex’, ‘Petr’]

# От начала списка до элемента с индексом 2 включительно
Names [:2] # Результат [‘Ivan’, Maria’]

# От начала до конца с шагом 2 (каждый второй элемент будет пропущен)
Names [::2] # Результат [‘Ivan’, ‘Alex’]
```

Списки (массивы)

- Удаление элементов из списка по индексу осуществляется с помощью метода **pop().** При этом удаляемый из списка элемент возвращается пользователю (то есть, этот элемент можно использовать). Если индекс в скобках не указан, то удаляется последний элемент списка.

Например:

```
student=Names.pop(2) #Результат student = ‘Alex’
                     #Names= [‘Ivan’, ‘Maria’, ‘Petr’]
```

- Возможно удалить элемент, непосредственно указав его, с помощью метода **remove()**.

Например:

```
Names.remove(‘Ivan’) #Результат  Names= [‘Maria’, ‘Petr’]
```

- Возможна вставка элемента в список с указанием индекса (позиции), куда он должен быть вставлен. Для этого используют метод **insert()**.

Например:

```
Names.insert(0, ‘Ivan’) #Результат  Names= [‘Ivan’, ‘Maria’, ‘Petr’]
```

Списки (массивы)

- **Копирование списка** осуществляется методом .copy(). Например:

CopyNames = Names.copy() #Результат CopyNames= [‘Ivan’, ‘Maria’, ‘Petr’]

- **Объединение** двух списков проще всего реализовать через операцию ‘+’. Например:

All = Names + CopyNames

#Результат All= [‘Ivan’, ‘Maria’, ‘Petr’, ‘Ivan’, ‘Maria’, ‘Petr’]

Другие полезные методы для работы со списками:

.clear() – удаление всех элементов списка (список остается пустым []);

.count() – подсчет количества элементов списка, совпадающего с заданным значением;

.extends() – добавить в конец списка заданные в параметрах набор элементов;

.index() – возвращает индекс первого элемента, совпадающего с заданным значением;

.reverse() – инверсия списка;

.sort() – сортировка списка.

Циклы

- Как правило циклы используются для перебора коллекций значений (например, списков). Общая конструкция выглядит следующим образом:

```
for <переменная> in <коллекция>:
    <действия>
```

Например:

```
for x in [1,2,3]:
   print (x*x)
```

#Результат 1 4 9

Вложенные конструкции

- Циклы и условия можно вкладывать друг в друга для получения более сложных программ. Общим критерием вложенности является отступы.

Например:

```
for x in [-1,2,-3]:
    if x<0:
      print (x*x)
    else:
      print (x)
#Результат 1 2 9
```

Ассоциативные типы данных (словари или дикты)

- Ассоциативный тип данных – это такая коллекция, где каждый элемент является парой **ключ-значение**. Для его создания нужно указать элементы внутри фигурных скобок. Синтаксис элемента в словаре: {ключ:значение}

- Ключами словаря могут быть строки и числа, а значениями почти что угодно – числа, строки, списки, даже другие словари! Например,

students = {‘Alex’:’19-VA1’} #Здесь ключ ‘Alex’ ассоциируется со значением ’19-VA1’

#Так мы установили связь студента с его группой

Теперь обращение `students ['Alex']` вернет значение ’19-VA1’

- Добавление новой пары ключ-значение:

students [‘Maria’] = ’19-VV3’

Теперь коллекция students содержит две пары: {‘Alex’:’19-VA1’, ‘Maria’:’19-VV3’}

Итерации словарей

- Такие коллекции могут перебираться (итерироваться) в цикле как по ключам, так и по значениям. Например,

```
for name in students.keys():
    print (name)
```

#Результат: ‘Alex’, ‘Maria’

```
for group in students.values():
    print (group)
```

#Результат: ’19-VA1’, ’19-VV3’