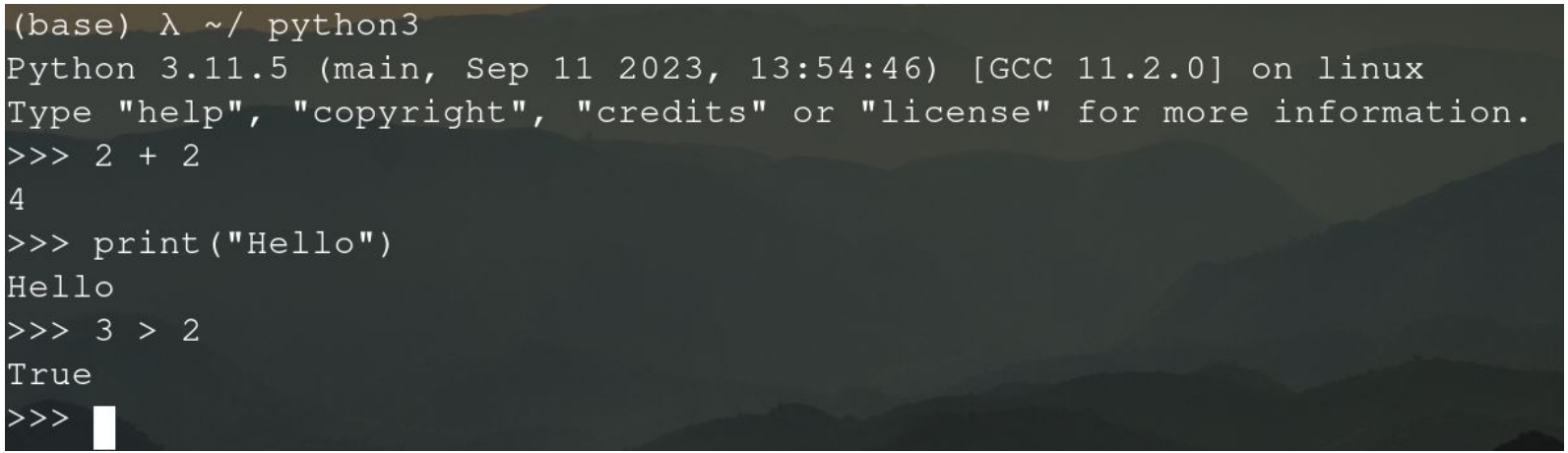
1. Python Into

Syllabus

- Начнем с основ python и минимально требуемой математики
- После этого непосредственно перейдем к методам машинного обучения
- Занятия будут проходить раз в неделю и состоять из лекционной части и практической
- Перед каждой лекцией будет небольшой опрос по предыдущей теме
- По ходу курса будет несколько лабораторных работ, требующих самостоятельной работы по пройденным темам

Python

- высокоуровневый *интерпретируемый* язык программирования общего назначения с *динамической строгой* типизацией и *автоматическим управлением памятью*



Installation

- Скачать с http://python.org/download/
- Для python доступно огромное количество библиотек для решения широкого спектра задач. Установить их можно при помощи pip (“Pip Installs Packages”), который вызывается python -m pip install …
- .py - расширение исполняемых файлов python

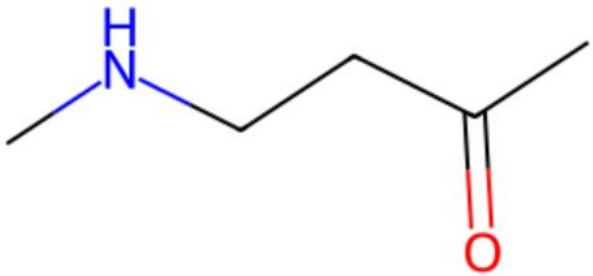
Jupyter Notebook

- Удобный способ работать с python - jupyter notebook
- Ноутбук состоит из ячеек с кодом, которые можно последовательно запускать
- Есть возможность пользоваться ноутбуками в облаке - google colab (https://colab.research.google.com)

```
[1]: from rdkit import Chem

[2]: Chem.MolFromSmiles("CC(=O)CCNC")
```

[2]:



Types

- bool - True или False
- int - целые числа
- float - числа с плавающей точкой
- str - строки
- list - список (динамический массив)
- tuple - кортеж (неизменяемый)
- dict - словарь (хранит данные в формате ключ - значение, ключ должен быть хэшируемый)
- set - множество (хранит только уникальные элементы, позволяет быстро отвечать на вопросы “есть ли x в множестве A?”)

List

```
a = list()
a = []

a.append(1)
a.append(5)
print(a)
# [1, 5]

a.extend([3, 4])
print(a)
# [1, 5, 3, 4]

a[3] = -1
print(a)
# [1, 5, 3, -1]

print(len(a))
# 4
```

Tuple

```
a = (1, 2, 3)
print(type(a))
# <class 'tuple'>

a[0] = -1
# TypeError: 'tuple' object does not support item
assignment
```

Dict

```
a = dict()
a = {}

a["alice"] = 10
a["bob"] = 3
print(a)
# {'alice': 10, 'bob': 3}

print("alice" in a)
# True
print("tom" in a)
# False

print(a.keys())
# dict_keys(['alice', 'bob'])
print(a.values())
# dict_values([10, 3])
```

Set

```
a = set()
a = {1, 2, 2, 3}
print(a)
# {1, 2, 3}

a.add(10)
print(a)
# {10, 1, 2, 3}

a.add(1)
print(a)
# {10, 1, 2, 3}
```

If/Else

```
a = 3
if a > 3:
    print("a > 3")
elif a >= 2:
    print("a in [2; 3]")
else:
    print("a < 2")
```

For/While loops

```
a = [1, 2, 3, 4, 5]

for idx in range(len(a)):
    print("a_i * 2 is ", a[idx] * 2)

for current_element in a:
    print("a_i * 2 is ", current_element)

idx = 0
while idx < len(a):
    print("a_i * 2 is ", a[idx] * 2)
    idx += 1
```

Functions

```
# Bad way
def say_hello(name):
    print("Hello, " + name + "!")

# Good way
def say_hello(name : str) -> None:
    print("Hello, " + name + "!")

def is_odd(x : int) -> bool:
    if x % 2 == 1:
        return True
    else:
        return False

def is_odd(x : int) -> bool:
    return x%2 == 1

say_hello("Ivan")
```

Recursion

```
def fact(n : int) -> int:
    if n <= 1:
        return 1
    return n * fact(n - 1)

print(fact(6))
# 720

def fibbonacci(n : int) -> int:
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibbonacci(n - 1) + fibbonacci(n - 2)

print(fibbonacci(7))
# 13
```

Map + lambda

```
def square(x : float) -> float:
    return x * x

a = [1, 2, 3, 4, 5]

b = list(map(square, a))
print(b)
# [1, 4, 9, 16, 25]

b = list(map(lambda x: x*x, a))
print(b)
# [1, 4, 9, 16, 25]
```

Classes

```
class Animal:

    def __init__(self, name : str) -> None:
        self.animal_name = name

    def print_name(self) -> None:
        print(self.animal_name)

    def get_info(self) -> str:
        return "Name: " + self.animal_name

animal = Animal("Buddy") # or Animal(name="Buddy")
print(animal.get_info())
# Name: Buddy

animal.print_name()
# Buddy
```

Inheritance

```
class Animal:

    def __init__(self, name : str) ->  None:
        self.animal_name = name

    def print_name(self) ->  None:
        print( self.animal_name)

    def get_info(self) -> str:
        return "Name: " + self.animal_name

class Dog(Animal):

    def get_info(self) -> str:
        return "Type: Dog\nName: "  + self.animal_name

general_animal = Animal(name= "Buddy")
print(general_animal.get_info())
# Name: Buddy

buddy = Dog(name= "Buddy")
buddy.print_name()
# Buddy

print(buddy.get_info())
# Type: Dog
# Name: Buddy
```

Magic methods

- __init__ - вызывается при создании объекта
- __call__ - вызывается при обращении к объекту через ()

```
class Foo:

    def __init__(self, name="Bob"):
        self.name = name
    def __call__(self):
        print(self.name)


foo = Foo()
foo()
```

- __repr__ - метод, возвращающий информативное описание объекта в виде строки

f-strings

```
print(f"2 + 2 is {2+2}")
# 2 + 2 is 4

a = "Alice"
print(f"Hello, {a}")
# Hello, Alice

ans = 1/3
print(ans)
print(f"{ans:.2f}")
# 0.3333333333333333
# 0.33
```

Перерыв

Практика