Машинное обучение в финансах

Лекция 3: Введение в программирование на Python. Numpy. Matplotlib

Роман В. Литвинов*

*CRO
Финансовая Группа БКС

Высшая школа экономики, Март 2021

Содержание

Содержание

Disclaimer: стану ли я первоклассным программистом?

Краткий ответ - нет. Об этом надо помнить... и не строить иллюзий. Для того, чтобы стать хорошим программистом вам потребуется несколько лет работы. В рамках курса такого объема, это невозможно.

Но... Не надо думать, что все, что мы будет делать бесполезно. К концу курса вас и человека не умеющего программировать должна будет разделять "пропасть".

NB: Представьте себе ситуацию, когда вы ребенок, еще не умеющий разговаривать. Попробуйте убедить взрослого сделать что-то более-менее сложное (алгоритм в два-три шага)... С машиной это не пройдет, на крики и размахивания руками реагировать она не будет.

Disclaimer: стану ли я первоклассным программистом?

Наши основные задачи:

- убрать коммуникативный барьер и наладить базовый диалог с машиной
- научиться писать (пусть простой, но свой) код на Python (вы получите колоссальное преимущество в виде нового мощного инструмента решения ваших задач)
- научиться понимать (в тч с помощью поисковика и литературы) код, написанный другим человеком. Научиться вносить изменения в этот код.
- научиться использовать библиотеки, созданные другими командами программистов.

Почему Python?

Основные преимущества:

- сравнительно легкий для изучения, но при этом мощный язык. Объектно-ориентированный. Понятный и простой синтаксис. Не настолько "жесток"к новичку (динамическая типизация). Интерпретируемый - код работает без компиляции.
- кросс-платформенный и бесплатный (по сравнению с Matlab, например).
- огромное количество существующих библиотек (математика, data science, ML).
- возможность создания интерактивных документов, содержащих исполняемый код, данные, графики, текст (в тч формате Latex) в Jupyter notebook.

Почему Python?

Минусы:

- работает медленее низкоуровневых языков, типа C/C++.
- не всегда эффективно использует память

Но эти минусы во многом могут быть нивелированы за счет использования специфических технологий и библиотек: Numpy, Numba, Cython, параллелизация, написание/ вызов библиотек и функций написанных на Fortran, C/C++ и проч.

Содержание

Алгоритм

Давайте попробуем разбить на шаги выполнение какой-нибудь "бытовой"задачи. Например, пересечение улицы через регулируемый пешеходный переход.

Есть ли в таком "алгоритме"логические конструкции? (что-если) Циклы? Важна ли последовательность исполнения шагов? Продолжим разбивать множество шагов на еще более "мелкий"набор инструкций.

Алгоритм

Последовательность инструкций, каждая из которых описывает простую операцию. При этом в целом, такая последовательность решает определенную (сложную) задачу.

Чтобы компьютер мог выполнить ваш алгоритм, алгоритм должен быть написан на языке понятном компьютеру (языке программирования).

Переменные

На самом простом уровне (не вдаваясь в дебри объектно-ориентированной философии), переменная состоит из двух частей - имени переменной и ее значения.

Слева мы помещаем имя переменной затем ставим знак = и после этого присваиваем переменной значение.

```
x=5
my_var=3
```

В имени переменной можно использовать: строчные и прописные буквы (camel humps: MyVar), цифры (0-9), нижние подчеркивания (см. выше).

Нельзя использовать: зарезервированные слова (if, for, print. Google - python reserved words), начинать имя с цифры.

NB: Старайтесь использовать имена содержащие смысл!

Типы

Каждая переменная в Python имеет свой тип (type). В простом смысле, это означает, что переменная обладает определенными ("жесткими") свойствами, которые влияют на ее значение и то, как она может использоваться.

Основные типы переменных:

- целое число (int) (1,2,3...)
- число с плавающией точкой (float) (64-битная аппроксимация (!) действительного числа)
- комплексное число (complex) (c=2i+3j)
- строка (str) (var="Hello wold!")
- булева переменная (bool) (True или False, 1 или 0)

precise (int, str, bool) vs imprecise (float, complex) types

Если вы не уверены в типе переменной, всегда можно проверить его использовав встроенную функцию type(). Напр., type(MyVar)

Операторы

Операторы позволяют нам манипулировать нашими переменными - производить арифмитические операции, логическое сравнение и тп.

Бывают унарные, бинарные и тернарные операторы (в зависимости от того, сколько аргументов они используют - один, два или три)

Основные унарные операторы, которые нам пригодятся:

Минус (для числовых типов возвращает -x):

```
-x
```

Логическое отрицание (True становится False и наоборот):

```
not x
```

Операторы

Полезные бинарные операторы - арифметические операции:

Сложение (возвращает сумму):

```
x+y
```

Вычитание (возвращает разность):

```
x-y
```

Умножение:

```
x*y
```

Деление:

```
x/y
```

Возведение в степень:

```
x**y
```

Целочисленное деление:

```
x//y
```

Деление по модулю (возвращает остаток от деления на число):

```
x%y
```

Операторы

Полезные бинарные операторы - сравнение:

Равно (проверяет соблюдается ли равенство, возвр. True или False):

```
x==y
```

Не равно (проверяет, что x не равно y, возвр. True или False):

```
x!=y
```

Меньше (проверяет, что x меньше y, возвр. True или False):

```
x<y
```

Меньше или равно:

```
x<=y
```

Больше (проверяет, что x больше y, возвр. True или False):

```
x>y
```

Больше или равно:

```
x>=y
```

Операторы

Принадлежность (является ли x элементом y, возвр. True или False):

```
x in y
```

Принадлежность (не является ли x элементом y, возвр. True или False):

```
x not in y
```

Тождественность (True, если x и y ссылаются на одно и тоже значение в памяти):

```
x is y
```

Не тождественность (True, если x и y не ссылаются на одно и тоже значение в памяти): :

```
x is not y
```

Полезные бинарные операторы - логические:

Логическое "и"(True, если значения x и y True, иначе False):

```
x and y
```

Логическое "или"(True, если значения x или y True, иначе False):

```
x or y
```

Выражения (expressions). Python как калькулятор

Большинство перечисленных нами операторов можно компоновать между собой (для правильной очередности операций можно использовать скобки, как в привычной нам арифметике - BIDMAS).
Пример:

```
x = (242+369)-48
```

Выражение (expression) - это код, состоящий из цепочки/комбинации значений, переменных, операторов, возможно функций, который вычисляет и возвращает определенное значение.

Пример (чуть посложнее, чем выше):

```
# вычисление стоимости бескупонной облигации
face_val = 100
t = 8
ytm = 0.06
zero_coupon_bond_val=face_val/(1+ytm)**t
```

NB: важность комментариев (для себя и команды)

Содержание

Функция

Как и в математике, функция в Python, получает на входе несколько аргументов и возвращает значение функции.

Кроме большого количества функций написанных профессиональными программистами, которые вы можете успешно использовать, вы также можете создавать свои функции. Это очень удобно для автоматизации вычислений, которые вам интересны.

Функция в Python определяется с помощью ключевого слова def, затем следует имя этой функции, в скобках перечисляются ее аргументы и ставится двоеточие.

```
def <name>():
    <body>
```

После этого следует блок кода (выделенный отступами), который отвечает за проведение соответствующих вычислений и возвращение результата.

Функция

В виде функции, пример вычисления бескупонной облигации, описанный нами выше может выглядеть следующим образом:

```
def zero_coupon_bond_val(face_val,t,ytm):
    # Функция, вычисляющая стоимость бескупонной
    # облигации для заданной номинальной
    # стоимости (face_val), срока (t, лет) и
    # доходности к погашению (ytm,
    # в виде десятичной дроби).
    zero_coupon_bond_val=face_val/(1+ytm)**t
    return zero_coupon_bond_val
```

Давайте теперь обсудим процесс написания подобной функции для вычисления, например, Евклидова расстояния между двумя точками на плоскости $\mathbb{R}^2$.

Функция

Здесь пример результата:

```
def euclid_dist(x1,x2,y1,y2):
    # Функция, вычисляющая Евклидово расстояние
    # между двумя точками с заданными
    # координатами.

    d1=(x1-y1)**2
    d2=(x2-y2)**2

    euclid_dist=(d1+d2)**0.5

    return euclid_dist
```

NB: syntactic sugar.

Содержание

Модуль/библиотека

Модуль представляет собой файл, содержащий код Python, в тч готовые (специализированные) функции (на самом простом уровне, это файл с расширением .py).

Коллекция из нескольких модулей на определенную тему называется пакетом или библиотекой (на практике, часто слова модуль, библиотека и пакет используются как взаимозаменяемые по смыслу).

Вы можете писать как свои модули (набор сходных по смыслу функций), так и использовать сторонние библиотеки (написанные другими программистами).

Это очень удобно. Вы импортируете модуль в свою программу и используете функци, содержащиеся в нем, для вычислений. Похоже на научный прогресс - не нужно каждый раз "изобретать велосипед". Можно взять уже проверенные временем и научным сообществом результаты/теоремы и двигаться вперед.

Модуль/библиотека

Полезные модули, которые мы будем часто использовать:

- NumPy (базовые математические функции, генераторы случайных чисел, линейная алгебра - матричные и векторные вычисления)
- SciPy (широкий спектр математических функций, численное интегрирование, интерполяция, оптимизация, статистика)
- Matplotlib (стандарт де-факто для построения двумерных и трехмерных графиков и визуализации результатов вычислений)
- Pandas (мощный пакет для анализа табличных данных и временых рядов)
- Scikit-learn (анализ данных и машинное обучение)

Модуль/библиотека

Как использовать модуль:

```
import <module>
```

или

```
import <module> as <name>
```

Пример для NumPy:

```
import numpy as np
```

Теперь вы можете использовать все функции содержащиеся в модуле:

```
np.sqrt(9)
np.sin(0.5)
np.exp(3)
```

Содержание

Основные структуры: списки, кортежи

Немного о строках и индексировании.

Как мы уже говорили раньше, строка (string) это один из базовых типов Pyton - str. Представляет из себя последовательность символов - букв, чисел, знаков (печатный текст, как простая аналогия).

Задается строка следующим способом (с помощью двойных или одинарных кавычек):

```
x = "Hello world!"
```

Каждый элемент строки имеет соотвествующий индекс (свой порядковый номер). Это похоже на индексирование элементов вектора. Доступ к соответствующему элементу строки можно получить с помощью его индекса (в тч отрицательного):

```
y=x[1], z=[-2]
```

NB: Важно помнить, что индексирование в Python начинается с нуля!

Основные структуры: списки, кортежи

Slicing. Мы также можем получить доступ к подмножеству элементов строки с помощью среза (slice)

```
a = x[0:5] # "Hello"
```

Работает срез по правилу:

```
data[start:stop)
```

Максимально широко можно использовать возможности среза следующим образом (шаг также может принимать отрицательные значения - поэксперименитируйте со значением -1):

```
data[start:stop:step]
```

Основные структуры: списки, кортежи

Для эффективных операций с данными в Python имеются встроенные структуры данных/контейнеры. Это "жесткие"схемы организации данных (состоящих из базовых типов - строк, чисел и тп в качестве элементов) со своими "встроенными"методами/функциями.

Важная концепция которая связана со структурами данных - это изменяемость (mutability). Есть структуры данных, которые после их создания невозможно изменить по ходу программы - неизменяемые (immutable), а есть те, в которые можно легко вносить изменения (стирать, модифицировать и добавлять новые элементы) - изменяемые (mutable).

Мы познакомимся сегодня с такими базовыми структурами как списки (lists), кортежи (tuples) и массивы NumPy (arrays) . Словари (dictionaries) и множества (sets), будут темами для самостоятельного изучения.

Основные структуры: списки, кортежи

Списки представляют одномерный (вектор-like) упорядоченный контейнер, элементами которого могут являтся различные объекты стандартных типов Python. Списки изменяемы (mutable).

Задаются списки при помощи квадратных скобок, а элементы перечисляются через запятую.

```
x=[1,7]
y=[12, 48, "Hello world!", "R"]
z=[[1,2],[3,4]]
```

У списков есть много полезных встроеных методов (функций), я продемонстирую вам несколько:

```
x.append(10) # добавление в конец списка элемента 10
x.extend(13,17,21) # нескольких элементов
```

Как и строки списки поддерживают индексирование:

```
y[2] # ??
x[0] = "start"
```

Основные структуры: списки, кортежи

Вы можете удалять элементы из списка:

```
del y[2:]
```

И склеивать списки:

```
x+y
```

Кортежи (tuples) похожи на списки, но являются неизменяемыми (immutable) контейнерами. Задаются с помощью круглых скобок (). Их можно склеивать, но нельзя изменять их содержимое. Как следствие, у них нет методов append и extend.

```
a=(1,2,3,4)
a[1]
b = ("Hi", "there")
a+b
```

Содержание

NumPy. Операции над массивами, векторизированные вычисления

Как уже говорилось выше NumPy представляет собой мощную математическую библиотеку. Для нас не менее важно, что NumPy имеет встроенную высокоэффективную структуру данных для операций с числами/численными массивами - массивы NumPy (arrays).

Нашей основной задачей будет работа с большими массивами данных состоящими из чисел (матрица, вектор, тензор и тп) . Думаю вы уже заметили, что списки или кортежи не очень удобны для этого.

Поскольку массивы NumPy могут состоять только из чисел одного типа, это позволяет проводить эффективно численные операции с таким массивом (не опасаясь нарваться, например, на строку в процессе вычислений)

NumPy. Операции над массивами, векторизированные вычисления

При условии, что библиотека Numpy загружена как np массив задается с помощью команды np.array():

```
#вектор-строка из четырех элементов
a=np.array([3.41, 5.2, 8, 9])
#вектор-столбец из четырех элементов
b = np.array([[3.41],[5.2], [8], [9]])
#матрица 2X3
m=np.array([[1,2,3],[4,5,6]])
```

Иногда полезно создать массив состоящий целиком изу нулей или случайных чисел:

```
#матрица 3X5 состоящая из нулей
A=np.zeros((3,5))
#вектор-строка состоящий из случайных чисел U(0,1)
R=np.random.random([1],7])
```

NumPy. Операции над массивами, векторизированные вычисления

Полезные команды NumPy связанные с массивами:

```
#возвращает размер массива
A.shape
#возвращает количество элементов в массиве
A.size
#создает массив чисел в формате (start,end,increment)
# end не включено!
np.arange(1,10,2)
#создает массив в формате (start,end,n_steps)
np.linspace(1,10,20)
#горизонтальная склейка массивов
#(одинаковое кол-во строк)
np.hstack([x,y])
#вертикальная склейка (одинаковое кол-во столбцов)
np.vstack([x,y])
```

NumPy. Операции над массивами, векторизированные вычисления

Доступ к элементам массива осуществляется с помощью индексов как и в работе со строками. Срезы работают аналогично [start:end:stride].

```
m[0,1] #первая строка, второй элемент
#возвращает количество элементов в массиве
m[0] #первая строка
```

Важно: NumPy гораздо эффективнее нежели цикл for производит математические операции (взятие квадратного корня, сумма столбца или строки) поскольку применяет операцию к массиву в целом, а не к скаляру (отдельному числу) (векторизированные вычисления).

Содержание

Циклы (for, while)

Цикл (loop) в Python это часть кода, которая позволяет повторять выполнение определенных действий. В Python имеется два типа циклов - for и while.

В общем виде формат использования цикла for выглядит следующим образом:

```
for <iter_var> in <sequence>:
    <loop body>
```

Теперь рассмотрим более конкретный пример:

```
for i in range(1,11,1):
    print(i**2)
```

NB: блок кода, выполняемого в цикле, выделяется отступом.

Циклы (for, while)

Цикл while сочетает в себе условную конструкцию и выполнение цикличной операции. Такая операция будет выполнятся до тех пор, пока будет верным (True) заданное нами условие.

Общий формат использования цикла while:

```
while <condition>:
    <loop body>
```

Более конкретный пример:

```
i=10
while i >=3:
    print(i**2)
    i-=1
```

NB: i-=1 то же самое, что и i=i-1 (i+=1 это i=i+1).

Циклы (for, while)

Бонус - comprehension

Содержание

Условные конструкции (if statements)

Условные конструкции позволяют нам выполнять действие/действия в случае, если соблюдаются одно или несколько условий. Есть несколько типов подобных конструкций: if, if-else, if-elif.

Общий формат использования if:

```
if <condition>:
    <body>
```

Более конкретный пример:

```
a,b =2,3
if a<b:
    print(a+b)
```

Можно проверять выполнение нескольких условий:

```
a,b =2,3
if a<b and a!=0:
    print(b/a)
```

Условные конструкции (if statements)

Комбинация из if-else используется в случае, когда необходимо выполнить один блок кода при соблюдении условия и другой в противном случае.

Общий формат использования if-else:

```
if <condition>:
    <body>
else:
    <body>
```

Более конкретный пример:

```
a,b = 2, 3
if a!=0:
    print(b/a)
else:
    print("На ноль делить нельзя!")
```

Содержание

Matplotlib. Визуализация

Как уже говорилось ранее Matplotlib представляет собой мощную библиотеку для визуализации результатов вычислений.

Большой набор примеров графиков, которые можно получить с помощью Matplotlib (галерея) находится по адресу: `https://matplotlib.org/stable/gallery/index.html`

Давайте рассмотрим пример кода для построения графика, который мы видели во второй лекции - зависимость пройденного пути от времени.

Это довольно простой пример, который, с теми или иными вариациями, мы будем использовать довольно часто по ходу курса. Я прокомментирую все, что происходит на каждом шаге выполнения кода.

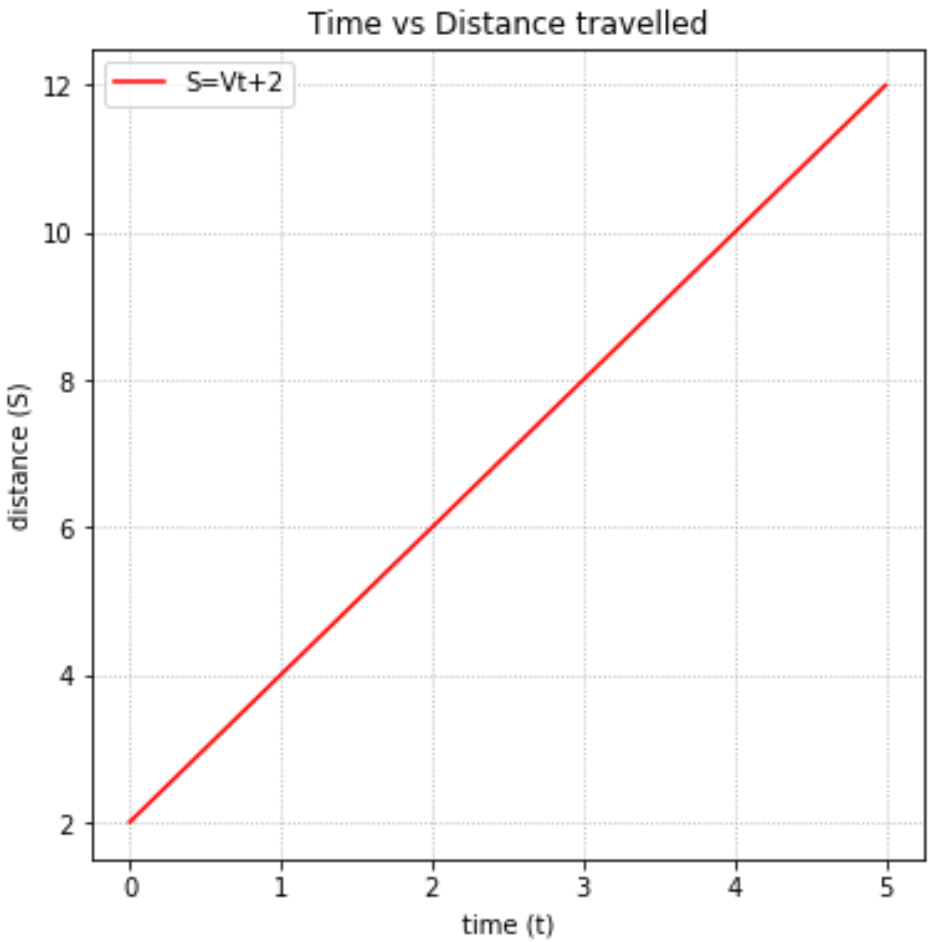
Matplotlib. Визуализация

```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0,5,100)
S = 2*t+2

fig,ax=plt.subplots(figsize=(6,6))
plt.plot(t, S, '-r', label='S=Vt+2')
plt.title('Time vs Distance travelled')
plt.xlabel('time (t)')
plt.ylabel('distance (S)')
plt.legend(loc='upper left')
plt.grid(visible=True, linestyle=':')
plt.show()
```

Matplotlib. Визуализация



Matplotlib. Визуализация

Попробуйте поэксперементировать с параметрами графика - цветами и форматом линий, количеством кривых, видимостью сеток для области построения и тп.

Для этого есть смысл почитатть документацию пакета Matplotlib и посмотреть какие аргументы (параметры) доступны у таких методов как plot(), grid() и тп.

`https://matplotlib.org/stable/api/pyplot_summary.html`

Другие полезные библиотеки, которые мы иногда будем использовать для визуализации это bokeh (`https://bokeh.org`) и seaborn (`https://seaborn.pydata.org`). Советую вам с ними ознакомиться.

Мы будем обсуждать код с их использованием отдельно, но для вас важно научиться самостоятельно ориентироваться в новых пакетах (их возможности, доступные функции, галерея базовых примеров и тп).

Содержание

Литература

Дополнительная литература к сегодняшней лекции:

- McKinney, W. (2018). Python for data science. O'Reily. (есть на русском)
- Scopatz, A., Huff, K. (2015) Effective computation in physics. O'Reily.
- Pine, D. (2019) Introduction to Python for science and engineering. CRC Press.
- Kinder, J., Nelson, P. (2015) A student's guide to Python for physical modeling. Princeton University Press.