1. Знакомство с инструментами разработки

На этом занятии мы продолжим знакомство со средствами разработки, типами данных и операциями с ними.

1. Модули и пакеты

Никто в больших проектах не пишет код в один файл (хочется верить). Для организации кода в Пайтон придумали разделять код с помощью модулей и пакетов. Модуль в Python --- это обычный файл с расширением .py, который содержит определения переменных, функций, классов, и который также может содержать импорты других модулей. Импорт выполняется стандартной командой import.

Конструкция from mymodule import my_function позволяет импортировать только определённые объявления из модуля. Чтобы импортировать все объявления из модуля, пишут from mymodule import *.

Сейчас мы создадим свой модуль на основе простой программки, которую делали на прошлой практике. Заодно посмотрим, как в python организовать функции. Открываем консоль и переходим в директорию, в которой будем работать. Создаем модуль

```
>> atom mymodule.py
```

Он должен открыться как пустой файл в редакторе кода. Создаем две функции
// *Обсудить, что такое функции и зачем они нужны*

```
def compute_bmi(age, height, weight):
    if float(height) > 100:
        print('Что-то ты высоковат! Введи рост в метрах')
        height = input()
    bmi = float(weight) / (float(height)**2)
    print(f'Итак, тебе {age} лет, твой рост {height}, а весишь ты
{weight}')
    print(f'Твой индекс массы тела равен {bmi}')

    return bmi

def rate_bmi(bmi):
    if bmi<19:
        print('Нужно больше кушать!')
    elif (bmi>=19) and (bmi <= 25):
        print('Молодец! Так держать!')
    elif bmi > 25:
        print('Стоит сходить в зал или побегать')
    else:
        print('Что-то пошло не так')
```

Теперь открываем командную строку, запускаем интерпритатор и импортируем их:

```
>>> from mymodule import *
```

Мы импортировали обе функции из модуля, теперь можем с ними экспериментировать:

```
>>> bmi = compute_bmi(28, 1.7, 57)
```

```
Итак, тебе 28 лет, твой рост 1.7, а весишь ты 57
Твой индекс массы тела равен 19.723183391003463

>>> rate_bmi(bmi)
Молодец! Так держать!
```

При импорте мы можем подменять имена функций:

```
>>> from mymodule import rate_bmi as rate
>>> rate(bmi)
Молодец! Так держать!
>>> rate(19)
Молодец! Так держать!
>>>
```

На самом деле, когда Python импортирует модуль, выполняет все инструкции, которые в этом модуле на верхнем уровне определены, в том числе import. То есть можно импортировать модуль, который импортирует другой модуль, который … и т.д.

Давайте напишем игру «угадай число» и импортируем ее как модуль. Заодно на этом примере рассмотрим работу цикла while

```
import random

number = random.randint(0, 10)

while True:
    answer = input('Ваше число? ')
    if answer =='' or answer == 'exit':
        print('Выход')
        break

    if not answer.isdigit():
        print('Это не число!')
        continue

    answer = int(answer)

    if answer == number:
        print('Угадал!!!!')
        break
    elif answer < number:
        print('Бери больше')
    else:
        print('Многовато')
```

Оформите игру как функцию и импортируйте ее как модуль

Иногда модулей недостаточно, тогда используются более крупные объединения --- пакеты. Пакеты в Python --- это директория, которая содержит один или больше модулей.

Посмотрим, что лежит в стандартной библиотеке Пайтона.

```
import this
import inspect
inspect.getfile(this)
```

```
import os
os.listdir('C:\\Users\\Catring\\AppData\\Local\\Programs\\Python\\Pytho
n310\\lib').
```

Итак, мы научились создавать свои модули. Теперь давайте научимся пользоваться чужими.

2. *Утилита pip и виртуальное окружение*

Почему Python стал так популярен для научных вычислений? Сам по себе он не создавался в расчете на анализ данных или что-то вроде того. Это произошло, прежде всего, благодаря большой и активно развивающейся экосистеме пакетов, созданных сторонними разработчиками.

Где посмотреть, какие вообріще существуют пакеты? Существует «каталог пакетов python», лежит тут: https://pypi.org.

Как их устанавливать? Использую специальную утилиту: Package Installer for Python.

```
pip install название_пакета
```

Как мы уже проговорили, внутри между модулями и пакетами могут существовать зависимости. Из-за этого обновление пакета может сломать проект. Чтобы как-то от этого защититься, придумали «Виртуальное окружение». Давайте создадим его прежде, чем устанавливать сторонние библиотеки. Для начала создадим папку, в которой будем работать и перейдем в нее. Затем вводим команду

```
python -m venv env
```

Дл я активации виртуального окружения нужно запустить скрипт activate.bat, который лежит в Scripts

```
env\Scripts\activate.bat
```

Как проверить, что мы действительно находимся в виртуальном окружении?

```
>>> import sys
>>> sys.prefix == sys.base_prefix
False
```

Все, теперь можно безбоязненно устанавливать сторонние библиотеки, не боясь что-нибудь сломать в других своих проектах. Давайте же что-нибудь установим.

```
pip install requests
```

Просмотр деталей об установленном пакете:
`pip show название_пакета`
Список всех установленных пакетов:
`pip list`
Устаревшие версии пакетов
`pip list --outdated`
Обновление
`pip install package-name -upgrade`
Установка конкретной версии:

```
pip install package-name==1.0.0
```
Полностью переустановить пакет
```
pip install package-name --upgrade --force-reinstall
```
Полностью удалить пакет:
```
pip uninstall package-name
```

```
Например : pip install ruts
```

Давайте установим Джупитер ноутбукс

```
pip install jupyter
```

Запустим

```
jupyter-notebook
```

Самостоятельно

Найти корни уравнения $\boldsymbol{ax^2 + bx + c = 0}$

```
import cmath

a = 1
b = 2
c = 7

D = lambda: b ** 2 - 4 * a * c
d=D(a,b,c)

print(d)

x1 = (-b + cmath.sqrt(d)) / (2 * a)
x2 = (-b - cmath.sqrt(d)) / (2 * a)

print(x1)
print(x2)
```

Посмотрели, как удобно это делать в jupyter. *Хорошо бы сюда задачу интереснее*.

На примере вычисления дискриминанта рассмотрим, что такое лямбда-функции, зачем нужны и где их стоит использовать (конкретно здесь – синтаксический сахар).

Лямбда-функции могут иметь любое количество аргументов, но у каждой может быть только одно выражение. Выражение вычисляется и возвращается. Эти функции могут быть использованы везде, где требуется объект-функция.

Эта функция может иметь любое количество аргументов, но вычисляет и возвращает только одно значение

Лямбда-функции применимы везде, где требуются объекты-функции

Вы должны помнить, что синтаксически лямбда-функция ограничена, позволяет представить всего одно выражение

Также работает с логическими выражениями

```
max_number = lambda a, b: a if a > b else b
print(max_number(3, 5))
```

Чтобы деактивировать виртуальное окружение:

```
env\Scripts\deactivate.bat
```

Прим.: в той версии интерпритатора python, которую ставили мы с вами, утилита pip уже есть. На случай, если у вас ее по каким-то загадочным причинам не оказалось, необходимо скачать скрипт с https://bootstrap.pypa.io/pip/get-pip.py , запустить его.

3. Anaconda и Spyder

Дома вы должны были установить дистрибутив Anaconda. Это дистрибутив python и R, для научных вычислений, преумущественно в области data science. Он включает графический интерфейс Anaconda Navigator как альтернатива интерфейсу командной строки, а также более двух сотен предустановленных пакетов, ориентированных, преимущественно, на работу с данными.

Чем это лучше pip? Иначе реализовано управление зависимостями пакетов.

Откроем навигатор, посмотрим, что там есть. Посмотрим, как тут можно создавать виртуальное окружение, ставить, обновлять и откатывать пакеты.

Теперь открываем Spider.

Кратенько смотрим, чего хорошего есть в spyder

4. Коллекции

На прошлой неделе мы начали рассматривать типы данных в python. Теперь рассмотрим структуры данных, используемые в python. Python поддерживает так называемые коллекции.

Коллекция --- это переменная-контейнер, в которой может содержаться какое-то количество объектов, где объекты могут быть одного типа или разного.

В случае списков это упорядоченные наборы элементов, которые могут быть разных типов.

```
empty_list = []
empty_list = list()
none_list = [None] * 10
collections = ['list', 'tuple', 'dict', 'set']
user_data = [ ['Elena', 4.4], ['Andrey', 4.2] ]
```

Для получения длины списка вызывают встроенную функцию len(). В Python не нужно явно указывать размер списка или вручную выделять на него память. Размер списка хранится в структуре, с помощью которой реализован тип список, поэтому длина вычисляется за константное время

```
len(collections)
```

Чтобы обратиться к конкретному элементу списка, мы обращаемся к элементу по индексу. Нумерация элементов начинается с нуля.

```
print(collections)
print(collections[0])
print(collections[-1])
```

Можно использовать доступ по индексу для присваивания (изменения элементов):

```
collections[3] = 'frozenset'
print(collections)
```

С помощью оператора in можно проверить, существует ли какой-то объект в списке:

```
'tuple' in collections
```

Срезы в списках работают точно так же, как и в строках. Создадим список из 10 элементов с помощью встроенной функции range и поэкспериментируем на нём со срезам

```
range_list = list(range(10))
print(range_list)

range_list[1:3]

range_list[::2]

range_list[::-1]

range_list[5:1:-1]
```

Как и все коллекции, спсиски поддерживают протокол итерации --- мы можем итерироваться по элементам списка, используя цикл for.

```
collections = ['list', 'tuple', 'dict', 'set']
for collection in collections:
    print('Learning {}...'.format(collection)) # используем функцию format для форматирования строк
```

Обратите внимание, что итерация производится именно по элементам списка, а не по индексам, как во многих других языках. Часто бывает нужно получить индекс текущего элемента при итерации. Для этого можно использовать встроенную функцию enumerate, которая возвращает индекс и текущий элемент.

```
for idx, collection in enumerate(collections):
        print('#{} {}'.format(idx, collection))
```

Так как списки являются изменяемой структурой данных, мы можем добавлять и удалять элементы. Например, мы можем добавить в наш список collections элемент 'OrderedDict'.

```
collections.append('OrderedDict')
```

Если вам нужно расширить список другим списком, вы можете использовать метод extend, который добавляет переданный список в конец вашего списка

```
collections.extend(['ponyset', 'unicorndict'])
```

Также можно использовать перегруженный оператор +, который также добавляет переменную в конец вашего списка:

```
collections += [None]
```

Для удаление элемента из списка можно использовать ключевое слово del.

```
del collections[4]
```

Часто нам нужно найти минимальный/максимальный элемент в массиве или посчитать сумму всех элементов. Вы можете это сделать при помощи встроенных функций min, max, sum.

```
numbers = [4, 17, 19, 9, 2, 6, 10, 13]
print(min(numbers))
 print(max(numbers))
 print(sum(numbers))
```

Часто бывает полезно преобразовать список в строку, для этого можно использовать метод str.join():

```
tag_list = ['RTF', 'FAVT', 'EGF']
print(', '.join(tag_list))
```

Ещё одна часто встречающаяся операция со списками --- это сортировка. В Python существует несколько методов сортировки.

Больше не успели.