Занятие 6.
Python. Синтаксический сахар. Аннотации типов. Модули и пакеты

1

про python

python

Что такое python?

- высокоуровневый интерпретирумый язык с сильной динамической типизацией

Что значит интерпретируемый?

Комплириуемые языки:

- более высокая скорость исполнения, так как они конвертируются в машинный код
- для запуска нужно изначально скомпилировать программу, а затем запустить
- эти языки плотформо-зависимые
- например, C, C++

Интерпретируемые языки:

- выполняются интерпретатором построчно
- проверяет отступы, синтаксис и т д
- не проверяет типы (т к python язык с динамической типизацией), то есть ошибка деления на 0 возникнет на этапе выполнение, а не копиляции в байт код

Типизация - сильная, динамическая

Зачем?

- не нужно объявлять типы переменных при написании кода, python сам вычисляет их во время выполнения

Утиная типизация - если это выглядит как утра, плавает как утра и крякает как утка, то это, вероятно, и есть утка - то есть это определение **динамической типизации**

То есть важно то, какое поведение поддерживает объект, а конкретный тип неважен

```
a = 10 # не нужно писать int a = 10 как мы писали бы в C#
type(a) # вывести тип
```

```
int
```

```
isinstance(a, int)  # проверить, является ли переменная a типом int
```

```
True
```

Типизация - сильная, динамическая

Что значит сильная?

- Значит что язык не позволяет смешивать в выражениях различные типвы и не выполняет автоматические неявные преобразования

```
0.3 + 2
```

```
2.3
```

```
0.3 + "2"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 0.3 + "2"

TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

js - язык со слабой типизацией

```
%%js
alert(0.3 + 2) // выведет 2.3
```

```
%%js
alert(0.3 + "2") // выведет 0.32, то есть приведет 0.3 к строке
```

2

Синтаксический сахар

List comprehension (генератор списка)

упрощенный подход к созданию списка, может содержать в себе if-else

```
list_ = [1, 2, 3, 4]
new_list = [elem ** 2 for elem in list_]
print(new_list)
```

```
[1, 4, 9, 16]
```

```
# замена filter
nums = [1, 2, 3, 4, 5]
odd_nums = [n for n in nums if n%2 == 1]
print(odd_nums)
```

```
[1, 3, 5]
```

Dict comprehensions (генераторы словарей)

Зачем?

- для создания словарей

```
new_dict = {num: num**2 for num in range(1, 11)}
```

```
r1 = {
    'ios': '15.4',
    'ip': '10.255.0.1',
    'hostname': 'london_r1',
    'location': '21 New Globe Walk',
    'model': '4451',
    'vendor': 'Cisco'
}

lower_r1 = {key.lower(): value for key, value in r1.items()}
print(lower_r1)
```

```
{'ios': '15.4', 'ip': '10.255.0.1', 'hostname': 'london_r1',
```

Set comprehensions (генераторы множеств)

Зачем?

- для создания сетов

```
# получить уникальные числа
list_ = [1, 2, "3", 4, 3, 2, 1]
unique_nums = {int(num) for num in list_}
print(unique_nums)
```

```
{1, 2, 3, 4}
```

```
type(unique_nums)
```

```
set
```

3

Философия python

Дзен python

Красивое лучше, чем уродливое.
Явное лучше, чем неявное.
Простое лучше, чем сложное.
Сложное лучше, чем запутанное.
Плоское лучше, чем вложенное.
Разреженное лучше, чем плотное.
Читаемость имеет значение.
Особые случаи не настолько особые, чтобы нарушать правила.
При этом практичность важнее безупречности.
Ошибки никогда не должны замалчиваться.
Если они не замалчиваются явно.
Встретив двусмысленность, отбрось искушение угадать.
Должен существовать один и, желательно, только один очевидный способ сделать это.
Хотя он поначалу может быть и не очевиден, если вы не голландец [^1].
Сейчас лучше, чем никогда.
Хотя никогда зачастую лучше, чем прямо сейчас.
Если реализацию сложно объяснить — идея плоха.
Если реализацию легко объяснить — идея, возможно, хороша.
Пространства имён — отличная штука! Будем делать их больше!

PEP, PEP8

PEP

Python Enhancement Proposal - предложения по улучшению python

PEP index

PEP8 - соглашение о написании кода

ссылка на статью

Код читается намного чаще чем пишется, поэтому важно писать хорошо читаемый код

Всегда есть правильные вариации, важно выбрать единый стиль и придерживаться его

Документирование

```
def pow(a, b):
    """Функция возвращает результат возведения числа a в степень b."""
    return a ** b
```

```
help(pow)
```

```
Help on function pow in module __main__:

pow(a, b)
    Функция возвращает результат возведения числа a в степень b.
```

Как называть переменные и функции

```python
# плохо
def area(side1, side2):
    return side1 * side2

d = area(4, 5)
```

```python
# хорошо
def get_rectangle_area(length, width):
    return length * width


area = get_rectangle_area(4, 5)
```

Модули, пакеты и библиотеки

Модуль - файл с расширением .py

Модули предназначены для того, чтобы в них хранить часто используемые функции, классы, константы и т.п.

Пакет - папка, в которой есть __ init__.py и другие модули

Библиотека - тоже набор модулей, отвечает за набор задач, например, библиотека для отправки http запросов - requests, библиотека для работы с датами - datetime и тд

```
# как импортировать
import datetime
datetime.datetime.now()
```

```
datetime.datetime(2023, 4, 9, 21, 21, 3, 754877)
```

```
# или
from datetime import datetime
datetime.now()
```

```
datetime.datetime(2023, 4, 9, 21, 21, 30, 416876)
```

```
# импортирует все сразу, так лучше не делать!
from datetime import *
```

Аннотации

- позволяют обозначить тип принимаемых или возвращаемых значений, нужно для читаемости кода и в качестве самодокументации функции
- аннотации типов никак не обрабатываются при интерпретации кода
- аннотации типов помогают при написании кода в IDE, так как pycharm будет предлагать возможные функции для атрибута
- они могут обрабатываться статическими анализаторами типа mypy

```python
# без аннотаций
def get_full_name(first_name, last_name):
    full_name = first_name.title() + " " + last_name.title()
    return full_name
```

```python
# с аннотациями
def get_full_name(first_name: str, last_name: str) -> str:
    full_name = first_name.title() + " " + last_name.title()
    return full_name
```

3

Домашнее задание

Домашнее задание

1. Пройти тест (ссылка в чате)
2. Решать задачи из дополнительных материалов (ссылка в чате), минимум по 2 задачи из тем функции и декораторы

Спасибо за внимание!