

- комментарии

"""

Многострочные
комментарии

"""

- змеиный_регистр
- ВерблюжийРегистр

Динамическая типизация - тип переменной не задаётся явно, а определяется интерпретатором



Типы данных:

- bool
- set
- dict
- Sequence: string, tuple, list
- Numbers: int, float, complex

Классификация типов данных:

- Изменяемые
- Индексируемые
- Содержащие только уникальные элементы

Побитовые операции:

- & - Побитовое И
- | - Побитовое ИЛИ
- ~ - Побитовое НЕ
- ^ - Побитовое исключающее ИЛИ
- << - Побитовый сдвиг влево
-   • Побитовый сдвиг вправо

Операторы вхождения и тождественности:

- `in` - True, если значение входит в последовательность
- `not in` - True, если значение НЕ входит в последовательность
- `is` - True, если операторы идентичны (указывают на один объект)
- `not is` - True, , если операторы не идентичны

Индексы и срезы:

- Получение символа:

```
имя_переменной[индекс]
```

- Срез:

```
имя_переменной[начало:конец(не включительно):шаг]
```

`\` - экранирование символов

- `\n` - новая строка(перевод строки)
- `\r` - возврат каретки
- `\t` - Горизонтальная табуляция

07.09

Практика 1

Все переменные являются ссылками

Объект состоит из:

- id
- тип
- значение
- счётчик ссылок

Интерполяция и форматирование

Это ресурсозатратно:

```
name = "John"
age = 25
job = "developer"

name + ", " + str(age) + ", " + job
```

Лучше с помощью `.format()` или f-строк

```
# 1
"{name}, age, {}, {}".format(age, job, name=name)
# 2
f"{name}, age {age}, {job}"
```

Форматирование f-строк:

```
f"{name: <20}|{age:^10}|{job:*>30}"
```

Результат:

```
John | 25 | *****developer
```

09.09

Лекция 2. Структуры данных. Списки

Структуры данных:

- Списки (list)
- Кортежи (tuple)
- Словари (dict)
- Множества (set)

Списки служат, чтобы хранить объекты в определённом порядке; можно изменять в любой момент. Пример:

```
empty_list = []
new_empty_list = list()
list = [1, 2, 3]
zero = [0] * 5
# zero = [0, 0, 0, 0, 0]
a = [1, 2, 3] * 2
# a = [1, 2, 3, 1, 2, 3]
```

split() - разделяет строку по символу (по умолчанию - пробел) и возвращает списки

```
today = '09/09/2024'
list_date = today.split('/')
# list_date = ['09', '09', '2024']
```

У списков есть индексы

```
numbers = [1, '0', 23.]
print(numbers[1])
# '0'
print(numbers[-1])
# 23.0
print(numbers[0:1])
# [1, '0']
```

Вложенные списки

```
small_birds = ['golub', 'vorobey']
big_birds = ['eagle', 'straus']

all_birds = [small_birds, big_birds]
print(all_birds)
# [['golub', 'vorobey'], ['eagle', 'straus']]
```

Добавление элементов:

- `numbers.append(7)` - добавление значения в конец
- `numbers += 5` - добавление значения в конец
- `numbers.extend([23, 34])` - добавить к концу другой список
- `numbers.insert(2, 5)` - вставляет элемент в указанный индекс

Удаление элементов:

- `del numbers[3]` - удаляет по индексу
- `numbers.remove(8)` - удаляет по значения
- `numbers.clear()` - очищает индекс
- `numbers.pop(3)` - удаляет по индексу(по умолчанию последний) и возвращает удалённый элемент

Поиск элементов:

- `numbers.index(7)` - возвращает индекс элемента\
- `numbers.count(7)` - возвращает количество данного элемента\

Сортировка:

- `numbers.sort([key, reverse])` - сортирует список (метод)\
- `sorted(list, [key, reverse])` - возвращает отсортированный список (функция)
key задаёт функцию сортировки
reverse - если True, то в обратном порядке\

Функции списков:

- `len()`
- `min()`
- `max()`
- `sum()` \

Списочное включения

```
новый_список = [*операция* for *элемент списка* in *список*]
```

Пример:

```
old_prices = [120, 550, 410, 990]
discount = 0.15
new_prices = [int(product * (1 - discount)) for product in old_prices]
```

```
новый список = [*операция* for *элемент списка* in *список* if *условие*]
```

Пример:

```
numbers = [121, 544, 111, 99, 77]
number11 = [num for num in numbers if num % 11 == 0]
# [121, 99, 77]
```

Проверка на тип

```
isinstance(val, int):
```

Объединение списков попарно

```
list1 = ['a', 'b', 'c']
list2 = [1, 2, 3]
print(list(zip(list1, list2)))
```

```
[('a', 1), ('b', 2), ('c', 3)]
```

У for и while есть else

Выполняется, если `for` или `while` закончился без `break`

Моржовый оператор

`:=` - сравнивает условие и присваивает

Практика 2

```
# Бесконечность
float('inf')
```

```
# минус бесконечность  
float('-inf')
```

Лекция 3

Кортежи - неизменяемые списки

```
a = (1, 2, 3, 4, 5)  
b = 6, 7, 8  
c = (2,) # - кортеж из одного элемента  
print(type(a), type(b))
```

```
<class 'tuple'> <class 'tuple'>
```

Словарь - структура, в которой уникальные ключи связаны со значениями

Ключи должны быть неизменяемыми типами данных\

```
student_1 = {'group': 'K0709-23/1', 'age': 17}  
print(student_1['group'])
```

```
'K0709-23/1'
```

Метод `get()`

```
student_1 = {'group': 'K0709-23/1', 'age': 17}  
student_1.get('age')  
# 17  
student_1.get('course', 'no course')  
# 'no course'
```

Методы

- `keys()` - список ключей
- `values()` - список значений
- `items()` - список пар ключ, значение

Множество

Создаётся с помощью функции `set()`, работает как обычное математическое множество

- Неиндексируемый тип данных
 - Содержит только уникальные значения
- Операции со множествами:
- `set.union(other)` - объединение множества `set b other`
 - `set.intersection(other)`
 - ...
 - `frozenset()` - неизменяемое множество

match/case (что-то типо switch case, но с более сложной структурой)

```
match value:
    case 'load':
        print("Load...")
        load()
    case 'save':
        print("Save...")
        save()
    case _:
        default()
```

```
arr = [1, 2, 4, 5, 144, 344]
for i, val in enumerate(arr):
    print(i, val)
```

```
for i in (2**i for i in range(10)):
    print(i)
```

```
1
2
4
8
16
32
64
128
256
512
```

Лекция 4. Функции, пространство имён, обработок ошибок, модули

Функция:

```
def func_name(param):  
    pass
```

Функция - объект, такой же как и прочие объекты в Python

```
def add(a, b):  
    return a + b  
def subtract(a, b):  
    return a - b  
(subtract if a > b else add)(1, 2)
```

У функции также есть атрибуты, их можно посмотреть с помощью `__dict__`

```
def fun():  
    return 0  
  
fun.__dict__  
# {}  
fun.a  
# 10  
fun.__dict__  
# {a: 10}
```

Словарь атрибутов может быть использован для кеширования промежуточных значений..

Вложенные функции:

Внутренние функции не определены до тех пор пока не будет вызвана родительская функция

***lambda* функция** - Анонимная функция

Пространство имён - это раздел, внутри которого имя уникально и не связано с такими же именами в других пространствах имён

Области видимости:

- Область встроенных имён (при `import`)
- Глобальная область (глобальные переменные, доступные ТОЛЬКО для ЧТЕНИЯ в функциях)
- Локальная область (локальные переменные функции, НЕ доступные для ЧТЕНИЯ в глобальной области, доступна ТОЛЬКО для ЧТЕНИЯ во внутренних функциях)

`global` - переменная будет относиться к глобальной области\

`nonlocal` - переменная будет относиться к области выше

Замыкание - функция, которая запоминает значение из своей внешней области видимости, даже если эта область уже недоступна

```
def outer(x):
    def inner(y):
        return x + y
    closure = outer(10)
    print(closure(5))
# 15
```

Обработка ошибок

Исключение - код, который выполняется, когда происходит связанная с ним ошибка

```
short_list = [0, 1, 2]
pos = 3
try:
    short_list[pos]
except:
    print("pos > len(list)")
```

\

```
try:
    # code
except ValueError:
    print("...")
except IndexError as idxErr:
    print(idxErr)
except Exception as other:
    print(other)
else:
    # Код, который выполнится, если не было ошибок
finally:
    # Код, который выполнится всегда
```

Модули

Модуль - файл, содержащий код Python

Ссылка на внешний модуль осуществляется с помощью `import`, имя файла указывается без расширения `.py`

```
import math
print(math.pi)
```

```
\n\nfrom math import pi\nprint(pi)
```

Псевдонимы:

```
from random import randint as r
```

Список модулей системы: `pip list`

Свой модуль: просто файл в той же директории с расширением `.py`

Модуль может определить, выполняется ли он в основной области видимости, проверив свое собственное `__name__`, что позволяет использовать общую идиому для условного выполнения кода в модуле, когда он выполняется как сценарий или скрипт с параметром `python -m foo.py`, но не при импорте `import`:

```
if __name__ == "__main__":\n    main()
```

`__main__` - это имя среды, в которой выполняется код верхнего уровня. "Код верхнего уровня" - это первый указанный пользователем модуль Python, который начинает работать. Это "верхний уровень", т.к. он импортирует все остальные модули, необходимые программе. Иногда "код верхнего уровня" называют точкой входа в приложение.

args и kwargs

```
*args # Можно и другое слово\n**kwargs
```

```
def foo(*args):\n    print(f"{args =}")\n\nfoo(1, 2, 3, True, "fdd")
```

```
args = (1, 2, 3, True, 'fdd')
```

```
def foo(**kwargs):\n    print(f"{kwargs =}")\n\nfoo(a=1, b=3, c=True, d="JNdfkk")
```

```
kwargs = {'a': 1, 'b': 3, 'c': True, 'd': 'JNdfkk'}
```

```
def foo(pos1, *args, kw=1, **kwargs):  
    print(f"{pos1 =}, {args =}, {kw = }, {kwargs = }")  
  
foo("Pos", 1,2,3,4, kw2=2, kw3=3)
```

```
pos1 = 'Pos', args = (1, 2, 3, 4), kw = 1, kwargs = {'kw2': 2, 'kw3': 3}
```

```
def foo(pos1, /, *, kw=0)
```

- / - конец позиционных документов
- * - дальше идут только ключевые

Подсказки

Входные элементы

```
def foo(a: int): ...  
def foo(a: int = 1): ...  
def foo(a: int | float): ...  
def foo(a: int | float = 1): ...
```

Выходные элементы

```
def foo(a: int) -> int | float | str: ...
```

\

```
from typing import Union  
  
a: list[Union[int, float]] = [1]  
a
```

Лекция 5. Генераторы и декораторы

Функция считается генератором, если:

- Содержит одно или несколько выражений `yield` .
- При вызове возвращает объект типа `generator`, но не начнет выполнение.
- Методы `__iter()` и `__next()` реализуются автоматически.

- После каждого вызова функция приостанавливается, а управление передается вызывающей стороне.
- Локальные переменные и их состояния запоминаются между последовательными вызовами.
- Когда вычисления заканчиваются по какому то условию, автоматически вызывается StopIteration

Пример

```
def range_1(first=0, last=2, step=1):
    number = first
    while number < last:
        yield number
        number += step

ranger = range_1()
print(next(ranger))
print(next(ranger))
print(next(ranger))
```

Генераторные выражения

```
nums_squared_lc = [num**2 for num in range(5)]
print(type(nums_squared_lc))
# <class list>
nums_squared_gc = (num**2 for num in range(5))
print(type(nums_squared_gc))
# <class generator>
```

\

Генераторы хоть и дают существенное преимущество в объёме памяти, могут работать значительно медленнее, чем списки

Методы .send(), .throw(), .close()

- `.send()` - отправляет значение генератору
- `.throw()` - создаёт исключение (= raise)
- `.close()` - завершение выполнения генератора (= break)

Декораторы

Декораторы обвёртывают функцию, изменяя её поведение

`@wraps(func)` - нужно, чтобы выводилась информация про вызываемую функцию, а не информацию про декоратор

Note

zip()

```
for i in zip((1,2,3,4), ('a', 'b,', 'c', 'd')):  
    print(i)
```

(1, 'a') (2, 'b,') (3, 'c') (4, 'd')

Лекция 6. Работа с файлами, контекстные менеджеры, структурированные текстовые файлы

- Используется относительные и абсолютные пути

Для Windows:

- `path = r'C:\Users\...'`
- `path = 'C:\\Users\\...'`

Для Linux всё стандартно

`fp = open('filename')` - открытие файла

`fp = open('filename', mode='r')` - только для чтений, варианты mode:

- `r` - чтение
- `w` - запись
- `x` - эксклюзивное создание
- `a` - для добавления в конец
- - чтение + запись
- `t` - текстовый режим
- `b` \

Чтение

- `fp.read()` - Считать целиком, есть аргумент `size`
- `fp.readline()` - одну строку
- `fp.readlines()` - Возвращает список строк

Запись

- `fp.write()` - Всё записывается в одну строку, если мы сами не указываем `\n`
- `fp.writelines()` - Записывает в файл последовательность, также не добавляет `\n`

Контекстный менеджер

```
with open('filename') as fp:
```

```
with EXPRESSION as TARGET:  
    SUITE
```

Структурированные текстовые файлы

CSV - Comma-Separated Values

Значения, разделённые запятыми (или другими символами)

```
ID,Name,Date  
1,Jake,20.10.24  
2,John,19.10.24
```

Библиотека `csv` для python \

XML - eXtensible Markup Language

"Расширяемый язык разметки"

Библиотека `xml`

JSON - JavaScript Object Notation

Текстовый формат обмена данными, основанный на JS

```
{  
    "breakfast": {  
        hours: "08.00-10.00",  
        "items": {  
            "Eggs": "5.95",  
            "Pancakes": "6.00"  
        }  
    },  
    "launch": {  
        ...  
    }  
}
```

Библиотека `json`

Словарь в python с помощью функции `dumps()` переконвертировать в JSON, `loads` - обратно.

Лекция 7

Пакетный менеджер

`pip` - менеджер пакетов в python

- `pip --version` - Версия `pip`
- `pip list` - Список установленных пакетов
- `pip search <название>` - Поиск пакетов по названию или его части
- `pip show <название>` - Информация о пакете
- `pip freeze` - Список установленных пакетов в текстовом формате
- `pip install <название>` - Установка пакета
- `pip install -r requirements.txt` - Установка пакетов из `requirements.txt`
- `pip install <название>@<версия>` - Установка пакета с указанной версией
- `pip install --upgrade <название>` - Обновление пакета
- `pip uninstall <название>` - Удаление пакета
- `pip --help` - Помощь по командам `pip`

Виртуальное окружение

`pip` устанавливает пакеты глобально, что не очень хорошо.

Основная цель **виртуального окружения Python** - создание изолированной среды для python-проектов \

`venv` - встроенный модуль для создания виртуальных окружений

Создание виртуального окружения:

```
python -m venv /path/to/venv
```

Традиционно директория виртуального окружения называется `.venv` или `venv`

Активация виртуального окружения:

```
source .venv/bin/activate
```

Деактивация с помощью `deactivate` \

PEP8

Лекция 8. Отладка кода. Логирование

Отладка кода

Отладка кода - обнаружение, локализация и устранение ошибок \

- Отладка с помощью `print()`
- Отладка с помощью точек остановок
- Модули `pdb` или `ipdb`

По умолчанию в VS Code есть три точки остановки:

- Expression (выражение)
- Hit count (количество срабатываний)
- Log message (лог-сообщение)

Логирование

Модуль `logging` - позволяет вести логи как в консоле, так и в файле. Имеете уровни дебагинга: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`, `NOTSET` \