Уср питон михадюк милана 49 группа 1 курс

Урок 1

**Введение в Python**

* **Python** — интерпретируемый, объектно-ориентированный, высокоуровневый язык с динамической типизацией.
* Подходит для быстрого прототипирования, сценариев, веб-разработки, научных вычислений и др.

**Преимущества Python**

* Читаемый и лаконичный синтаксис.
* Высокая скорость разработки.
* Кроссплатформенность.
* Большая стандартная библиотека.
* Легкая интеграция с C/C++, Java, .NET, и др.
* Простой в освоении и приятный в использовании.

**История**

* Создан Гвидо ван Россумом в 1991 году.
* Название происходит от "Монти Пайтона", а не змеи.
* Развивается благодаря сообществу.

**Где используется**

Компании: Google, YouTube, Dropbox, Яндекс и др.  
Применения: GUI-программы, веб-приложения, научные задачи, базы данных, игры.

**Философия языка (Zen of Python)**

* "Красивое лучше, чем уродливое", "Простое лучше, чем сложное", "Читаемость важна" и др.

**Python 2.x vs 3.x**

* Python 3 — современная версия, но не полностью совместима с 2.x.

**Основы синтаксиса**

* Динамическая типизация: типы переменных не указываются.
* Чувствительность к регистру.
* Отступы важны (вместо фигурных скобок).
* Комментарии начинаются с #.

**Пример простой программы**

#!/usr/bin/env python

import sys

def main():

print('Hello there', sys.argv[1])

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Функции**

* Определяются с помощью def.
* Используют отступы.
* Поддержка строк документации (docstring).
* Локальные переменные.

**Работа со строками**

* Форматирование: format().
* Методы: lower(), upper(), strip(), find(), replace(), split(), join() и др.
* Проверки: isalnum(), isalpha(), isdigit().

**Условные операторы**

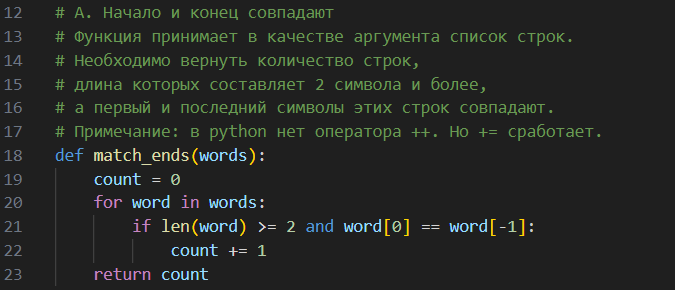
* Используются if, elif, else, отступы, логические операторы and, or, not.

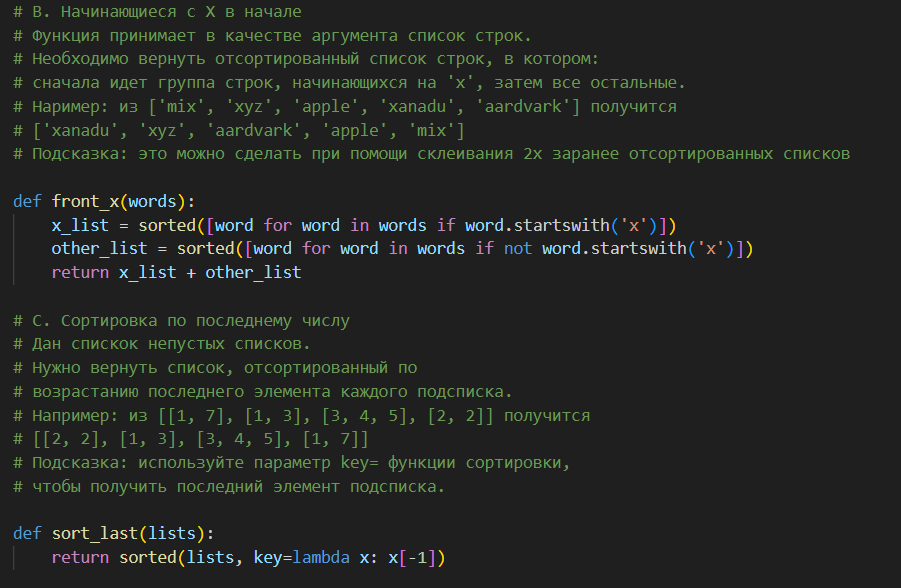
**Циклы**

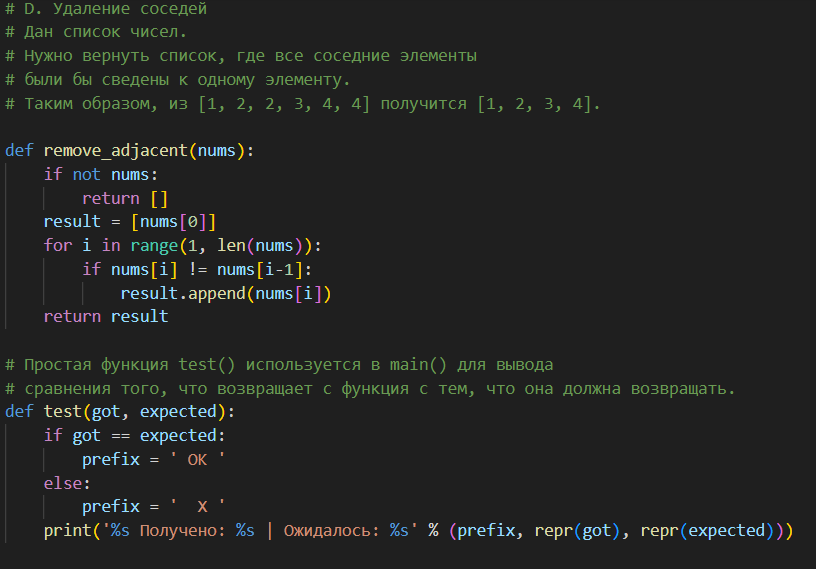
* for ... in позволяет проходить по элементам списка.
* Пример:

for num in [1, 4, 9, 16]:

print(num)









Урок 2. Типы данных

**Базовые типы данных в Python**

* **Числа**: int, float, complex, Decimal, Fraction
* **Строки**: 'spam', "text", b'byte'
* **Коллекции**: list, tuple, dict, set, frozenset
* **Прочее**: bool, None, файлы (open())

**Числа Decimal и Fraction**

* **Decimal**: точные десятичные числа, используются в финансовых приложениях, настраиваемая точность.
* **Fraction**: рациональные дроби вида numerator/denominator, поддержка преобразования из других чисел и строк.

**Кортежи (tuple)**

* Неизменяемые списки, защищены от изменений.
* Меньше по размеру, могут быть ключами словаря.
* Поддерживают те же операции, что и списки, кроме изменения.

**Словари (dict)**

* Коллекции пар "ключ-значение".
* Создаются разными способами: литералы, dict(), fromkeys(), генераторы.
* Основные методы: get(), items(), keys(), values(), pop(), update() и др.

**Множества (set, frozenset)**

* Наборы уникальных элементов, порядок не сохраняется.
* Операции: объединение (|), пересечение (&), разность (-), симметрическая разность (^).
* Методы: add(), remove(), discard(), clear(), copy(), update(), и др.
* frozenset — неизменяемое множество.

**Файлы**

* Работа через open(filename, mode, encoding).
* Режимы: 'r', 'w', 'a', 'b', 't', '+'.
* Методы: read(), readline(), readlines(), write(), writelines(), close(), flush(), seek().

**Генераторы**

* Компактный способ создания списков, множеств, словарей.
* Пример: [i for i in range(10)], {i: i+i for i in range(5)}, (i for i in range(5)) — возвращает генератор.

Эффективны по памяти, особенно в циклах и с функциями (sum(), max() и др.).  
  
  
Урок 3. Функциональное программирование в Python:

**Ввод данных**

* input("подсказка") — запрашивает ввод пользователя. Возвращает строку.

**Интерактивные циклы**

Цикл для ввода/вывода:

while True:

reply = input('Enter text:')

if reply == 'stop': break

print(reply.upper())

**Обработка ошибок**

**1. Проверка через isdigit():**

if not reply.isdigit():

print('Bad!' \* 8)

else:

print(int(reply)\*\*2)

**2. Обработка через try/except:**

try:

num = int(reply)

except:

print('Bad!' \* 8)

else:

print(num \*\* 2)

**Lambda (анонимные функции)**

* Краткая запись функций:

f = lambda x, y, z: x + y + z

* Можно использовать значения по умолчанию:

x = lambda a="fee", b="fie", c="foe": a + b + c

**map(), filter(), reduce()**

**map(func, seq)** — применяет функцию ко всем элементам:

list(map(lambda x: x + 3, [1, 2, 3]))

# [4, 5, 6]

**filter(func, seq)** — отбирает элементы, где функция возвращает True:

list(filter(lambda x: x > 0, range(-5, 5)))

# [1, 2, 3, 4]

**reduce(func, seq)** — сворачивает последовательность в одно значение:

from functools import reduce

reduce(lambda x, y: x + y, [1, 2, 3, 4])

# 10

**Урок 4«Системное программирование. Регулярные выражения**

**Регулярные выражения в Python**

* Используется модуль re.
* Основная функция: re.search(pat, str) — ищет первое совпадение по шаблону.
* Результат — объект match, у которого можно получить результат через match.group().

**Сырые строки**

* Использование префикса r'' удобно для регулярных выражений (не обрабатываются \).

**Основные шаблоны**

* Обычные символы (a, X, 9) и **метасимволы** (., \w, \d, ^, $, и т.д.).
* Классы символов:
  + \w — буквы, цифры и \_
  + \d — цифры
  + \s — пробельные символы
  + . — любой символ, кроме \n

**Повторы**

* \* — 0 или более раз
* + — 1 или более раз
* ? — 0 или 1 раз

**Наборы символов**

* [abc] — один из символов
* [^abc] — любой, кроме указанных
* [\w.-]+@[\w.-]+ — простейший шаблон email

**Группы**

* () — группировка, позволяет извлекать подстроки через group(n)
* (?:...) — группа без извлечения

**Поиск всех совпадений**

* re.findall() — возвращает список всех совпадений.
  + Если есть группы — список кортежей.

**Жадные и нежадные выражения**

* .\* — жадный (захватывает максимум)
* .\*? — нежадный (остановится на первом совпадении)

**Замена**

* re.sub(pat, repl, str) — замена совпадений на указанный текст.
* В repl можно использовать \1, \2 и т.п. для подстановки из групп.

**Компиляция выражений**

* re.compile(pat) — создаёт объект регулярного выражения для многократного использования.

**Работа с файловой системой**

Модули: os, os.path, shutil

* os.listdir(path) — список файлов в директории
* os.mkdir(), os.makedirs() — создать директорию(и)
* os.path.join(), abspath(), basename(), dirname(), exists()
* shutil.copy(), shutil.rmtree() — копирование и удаление файлов/директорий

**Работа с URL — urllib**

* urllib.request.urlopen(url) — открыть URL как файл
* read(), decode('utf-8') — чтение и декодирование содержимого
* urlretrieve(url, filename) — скачивание файла
* urllib.parse.urlencode() — кодирует словарь в строку запроса
* urllib.parse.urlparse() — разбирает URL на части

урок 5

**1. Модули**

* Программа в Python обычно состоит из главного файла и модулей (вспомогательных файлов с кодом).
* **Импорт модуля**: import module — загружает файл, компилирует его (при необходимости), создает объекты.
* **Импорт с выбором имен**: from module import name или from module import \*.
* **Пути поиска модулей**: определяются sys.path (домашний каталог, PYTHONPATH, стандартная библиотека и др.).
* Модули — это пространства имен, их атрибуты можно просмотреть через module.\_\_dict\_\_ или dir(module).
* **Атрибуты модуля**: \_\_name\_\_, \_\_doc\_\_, \_\_file\_\_, \_\_path\_\_, \_\_package\_\_.

**2. Пакеты**

* Пакет — это каталог с модулями, который становится пространством имен.
* Для корректной работы каталогов как пакетов они должны содержать файл \_\_init\_\_.py.
* Импортируется как: import dir1.dir2.mod.
* \_\_init\_\_.py запускается при первом импорте пакета, может содержать инициализационный код.

**3. Классы и объекты**

* Класс — это **шаблон (фабрика)** для создания экземпляров.
* Определяется с помощью class Name(Superclass):.
* Атрибуты создаются через присваивание, методы — через def внутри класса.
* Экземпляры создаются вызовом класса: obj = ClassName(...).
* Каждый экземпляр имеет своё пространство имен и может иметь уникальные данные.

**4. Наследование**

* Подклассы наследуют атрибуты и методы суперклассов.
* Синтаксис: class SubClass(SuperClass):.
* Интерпретатор ищет атрибуты сначала в экземпляре, потом в классе, затем в суперклассах.
* Метод \_\_init\_\_ можно переопределять и вызывать версию суперкласса вручную.

**5. Инкапсуляция**

* Принцип скрытия реализации: операции оформляются как методы, чтобы менять код в одном месте.
* Внутренние методы: начинаются с \_ (например, \_hidden\_method).
* Псевдоприватные атрибуты: \_\_attr — интерпретатор меняет имя, чтобы избежать конфликтов.

**6. Специальные методы**

* \_\_init\_\_: вызывается при создании экземпляра.
* \_\_str\_\_: отвечает за строковое представление объекта.
* Вызов методов: obj.method() ≈ Class.method(obj).

Урок 6: Декораторы, Исключения, Итераторы:

**1. Декораторы**

* Декораторы — функции, принимающие другую функцию и возвращающие модифицированную.
* Синтаксис:
* @decorator
* def func(): pass

Эквивалентно func = decorator(func)

* Декораторов может быть несколько, выполняются сверху вниз.
* Можно передавать параметры в декоратор: @pause(2)
* Применимы к методам классов, включая @staticmethod и @classmethod.

**2. Исключения**

* **Типы ошибок**:
  + *SyntaxError* — синтаксическая ошибка.
  + *Исключения (Exceptions)* — ошибки во время выполнения.
* **Обработка исключений**:
  + try/except — перехват и обработка.
  + else — выполняется, если ошибок не было.
  + finally — выполняется всегда.
* **Создание исключений**:
  + raise — возбуждение исключения вручную.
  + assert — проверка условий, вызывает исключение при ложности.
* **Пользовательские исключения** — наследуются от Exception.

**3. Итераторы и генераторы**

* **Итерируемые объекты**: поддерживают for...in, напр. списки, строки, файлы.
* **Итератор**: объект с методами \_\_iter\_\_() и \_\_next\_\_(). Конец — исключение StopIteration.
* **Генератор**:
  + Функция с yield, создающая значения "на лету".
  + Пример:
  + def fibonacci(max):
  + a, b = 1, 2
  + while a < max:
  + yield a
  + a, b = b, a + b

### Урок 7. Библиотеки

### Модуль itertools

Предоставляет функции для работы с итерациями.

* **Бесконечные последовательности:**
  + count(start=0, step=1) — бесконечный счётчик.
  + cycle(iterable) — бесконечно повторяет элементы.
  + repeat(elem, times=None) — повторяет элемент указанное число раз.
* **Контроль итераций:**
  + takewhile(pred, iterable) — возвращает элементы до первого ложного.
  + islice(iterable, start, stop, step) — срез итератора.
* **Комбинаторика:**
  + permutations(iterable, r) — перестановки r-элементов.
  + combinations(iterable, r) — комбинации без повторений.
  + product(\*iterables, repeat=1) — декартово произведение.
  + groupby(iterable, key=None) — группирует последовательные одинаковые элементы.
  + chain(\*iterables) — объединяет итерируемые объекты.

**Модуль collections**

Специализированные структуры данных.

* **Counter** — счётчик элементов:
  + Умеет считать, обновляться, получать наиболее частые элементы.
  + Поддерживает операции как с множествами (сложение, пересечение и т.д.).
* **OrderedDict** — словарь с сохранением порядка добавления:
  + Порядок влияет на сравнение.
  + Перемещение элемента в конец — move\_to\_end().
* **defaultdict(default\_factory)** — словарь с значением по умолчанию.
* **namedtuple(typename, fields)** — кортеж с именованными полями:
  + Методы: \_make(), \_replace(), \_asdict(), \_fields.

**Работа с CSV (csv модуль)**

Формат хранения табличных данных.

* **Чтение:**
  + csv.reader(file, dialect)
  + DictReader — чтение в словарь по заголовкам.
  + Автоопределение диалекта — csv.Sniffer().sniff().
* **Запись:**
  + csv.writer(file, dialect) — запись строк.
  + Диалекты (delimiter, quotechar, quoting, и др.).

**Работа с SQLite (sqlite3 модуль)**

Встраиваемая реляционная БД.

* Создание базы: sqlite3.connect('file.db').
* Создание таблиц — через conn.execute(SQL).
* Вставка, выборка, обновление данных:
  + Используется execute() и cursor().
  + Доступ к полям через Row-объекты.

**Работа с JSON (json модуль)**

Формат обмена данными.

* dumps(obj) — сериализация в JSON.
* loads(json\_str) — чтение из JSON.
* Поддерживает сортировку ключей, отступы, плотную упаковку.
* skipkeys=True — игнорирует неподдерживаемые ключи.

**Работа с XML (xml.etree.ElementTree)**

Формат структурированных данных.

* parse(file) — загрузка XML.
* Методы:
  + getroot() — корневой элемент.
  + find(), findall(), iter() — поиск.
  + set(), text, get() — доступ и изменение элементов.