ПМ2 Прикладное программирование.

РО 2.1. Создавать консольные приложения на Python.

Тема 1. Основы построения программ на Python.

Цель занятия:

Сформировать базовое представление о назначении языка Python, этапах его установки и запуске программ, а также научиться создавать простейшие программы с вводом, выводом данных и выполнением элементарных вычислений.

Учебные вопросы:

1. Назначение и области применения языка Python.
2. Установка Python и обзор инструментов разработки (IDLE, VS Code, терминал). Запуск программ.
3. Основные функции ввода-вывода.
4. Комментарии.

1. Назначение и области применения языка Python.

Назначение языка Python.

Python — это высокоуровневый, интерпретируемый язык программирования общего назначения.

Он был создан для того, чтобы упростить процесс разработки программ, сделать его более быстрым и понятным даже для начинающих.

Высокоуровневый язык — это язык программирования, который максимально скрывает детали работы компьютера и позволяет программисту сосредоточиться на логике программы.

Интерпретируемый язык — это язык, в котором программа выполняется построчно специальной программой — **интерпретатором**.

Особенности:

- нет этапа компиляции в полноценный исполняемый файл
- можно выполнять код сразу, без сборки
- удобно для обучения, тестирования, написания скриптов
- облегчает переносимость между платформами

Язык общего назначения — это язык, который не ограничен одной конкретной областью и позволяет писать программы самых разных типов.

Основные особенности языка:

- Простота и читаемость кода.
- Быстрая разработка.
- Кроссплатформенность.
- Расширяемость и большое сообщество.

Области применения Python:

- Веб-разработка. Серверная логика сайтов и веб-приложений. (Django)

  https://ru.hexlet.io/programs/python-django-developer
- Научные вычисления и анализ данных
- Машинное обучение и искусственный интеллект
- Автоматизация и скрипты
- Разработка игр
- Создание приложений и утилит

Вывод:

Python — универсальный язык, который сочетает простоту, мощность и широкие возможности.

Поэтому он используется в самых разных областях: от веб-сервисов и анализа данных до машинного обучения и автоматизации

2. Установка Python и обзор инструментов разработки. Запуск программ.

Установка Python.

1. Загрузка дистрибутива:

- Перейти на официальный сайт: python.org
- Открыть раздел Downloads
- Скачать версию для своей ОС (Windows, macOS, Linux)

www.python.org Download Python | Python.org Спросить

Python PSF Docs PyPI

python™

Donate

About Downloads Documentation Community Success Stor

Download the latest version for Windows

Download Python install manager

Or get the standalone installer for Python 3.14.0

Looking for Python with a different OS? Python for Windows, Linux/Unix, macOS, Android, other

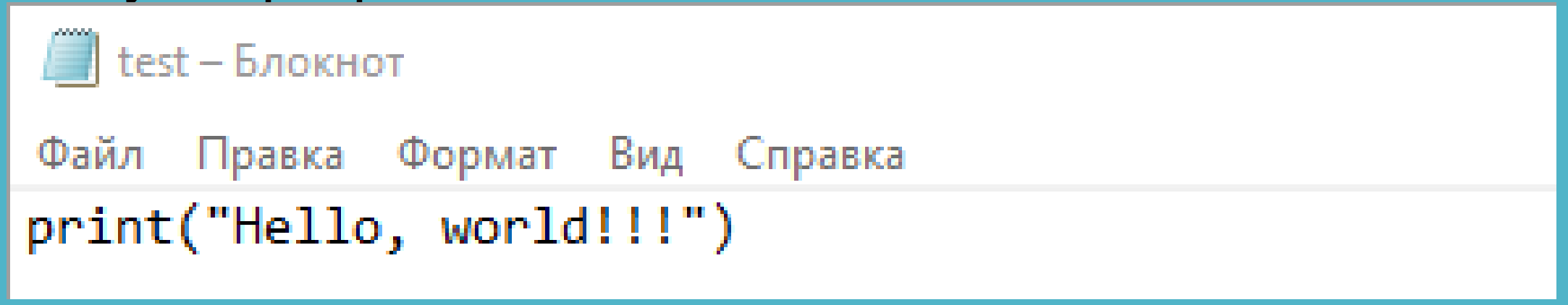
Want to help test development versions of Python 3.15? Pre-releases, Docker images

2. Установка (Windows):

- Запустить установщик
- Отметить галочку Add Python to PATH
- Нажать Install Now
- Проверить установку:

```
C:\Users\user\Documents>python --version
Python 3.14.0
```

Запуск программы:




```
C:\Users\user\Documents>python test.py
Hello, world!!!
```

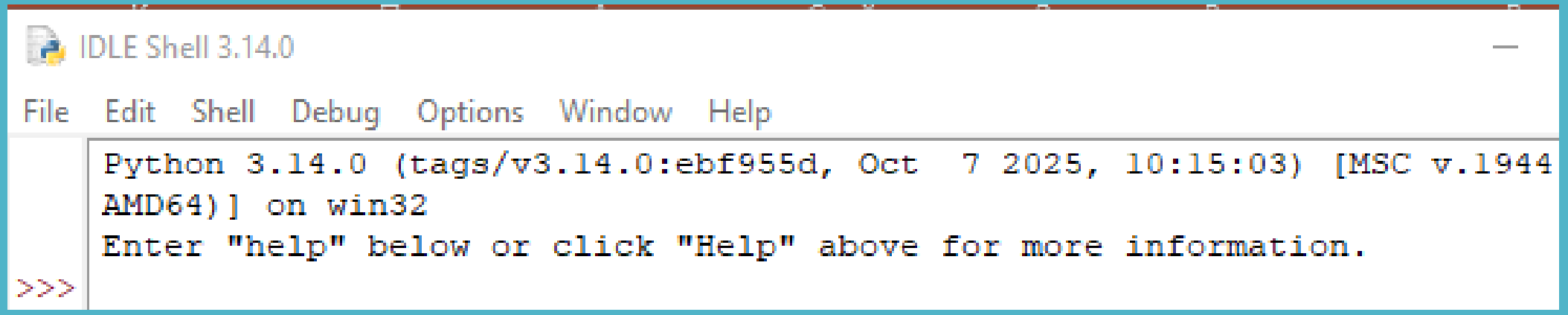
```
C:\Users\user>python
Python 3.14.0 (tags/v3.14.0:ebf955d, Oct  7 2025, 10:15:03) [MSC v.1944
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello")
Hello
>>>
```

Инструменты разработки Python

1. IDLE

Встроенная среда, устанавливается автоматически вместе с Python.

- Лёгкий интерфейс
- Встроенный Python Shell
- Ограниченные возможности
- Не подходит для больших проектов



2. Visual Studio Code (VS Code)

Универсальный редактор кода, широко используемый в обучении и профессиональной разработке.

- Расширение Python от Microsoft
- Автодополнение кода (IntelliSense)
- Отладка (debug)
- Интеграция с терминалом
- Работа с виртуальными окружениями
- Поддержка Git
- Поддержка Jupyter Notebook
- Бесплатный
- Подходит для большинства задач

3. PyCharm

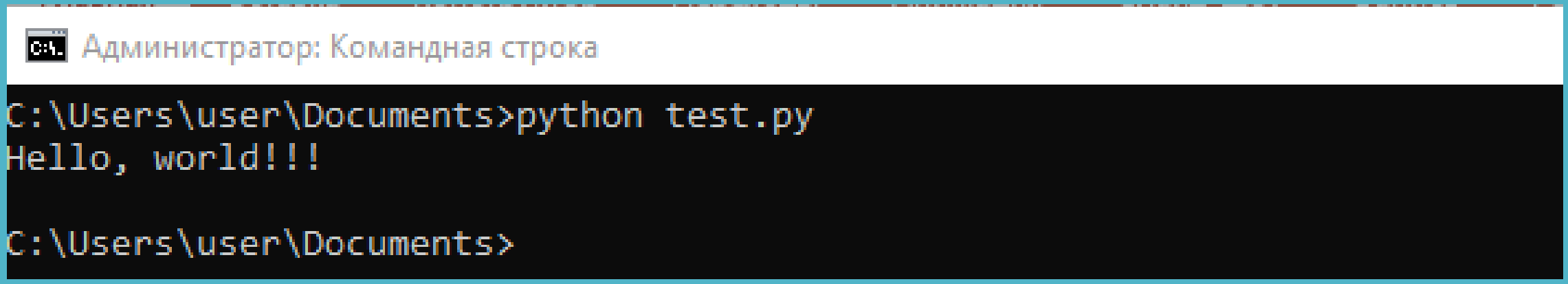
Одна из самых мощных и удобных профессиональных сред для Python от JetBrains.

- Очень качественная автодополняемость
- Удобная отладка
- Инструменты для Django, Flask
- Работа с Git, Docker
- Настройка виртуальных окружений
- Подсветка ошибок в процессе набора кода
- Профессиональная IDE
- Отлично подходит для крупных проектов
- Требовательна к ресурсам

4. Терминал (командная строка)

Один из базовых способов взаимодействия с Python.

Популярен в DevOps, на серверах, в автоматизации



5. Другие полезные инструменты.

Jupyter Notebook

- Формат интерактивных «тетрадей»
- Используется в анализе данных, ML, обучении
- Позволяет смешивать код, текст, графики

Thonny

- Не нужно скачивать Python отдельно
- Простая IDE, специально разработанная для новичков
- Поддерживает пошаговую отладку, удобный интерфейс

И множество других

3. Основные функции ввода-вывода.

Ввод и вывод — это основные способы взаимодействия программы с пользователем.

Вывод: отображение информации на экране.

Ввод: получение данных от пользователя.

В Python эти операции реализуются в первую очередь через функции **print()** и **input()**.

Функция **print()** — вывод данных на экран

Функция **print()** используется для отображения текста, чисел, результатов вычислений и любых других данных.

print(). Базовое использование:

```
1       print("Hello, Python!")
2       print(2 + 3)
```

```
Hello, Python!
5
```

Вывод нескольких значений (аргументов)

Можно вывести несколько элементов через запятую:

```
1    print("Результат:", 5 + 3)
2
```

```
C:\Users\user\PycharmProjects\PythonProject
Результат: 8
```

Python автоматически добавляет пробелы между элементами.

У функции **print()** есть аргументы по умолчанию.

Аргумент **sep** — разделитель

По умолчанию разделитель — пробел.

Его можно изменить:

```
1    print("A", "B", "C", sep="-")
2    print("A", "B", "C", sep="")
```

```
C:\Users\user\PycharmProjects\PythonProject\
A-B-C
ABC
```

Аргумент **end** — окончание строки

По умолчанию **print()** завершает строку символом новой строки (**\n**).

Можно изменить это поведение:

```
1    print("Hello", end=" ")
2    print("World")
```

```
C:\Users\user\PycharmProjects\PythonProject\
Hello World
```

Конкатенация (склеивание) строк в print()

Функция print() позволяет выводить несколько строк подряд и объединять их разными способами.

1. Конкатенация через запятую (рекомендуемый способ)

Если передать несколько аргументов через запятую, print() сам объединяет их, вставляя пробел по

```
print("Hello", "World")
```

```
Hello World
```

2. Конкатенация с помощью оператора +

Используется реже, потому что требует, чтобы все аргументы были **строками**.

```
print("Hello" + " " + "world")
```

```
Hello world
```

Функция input() — ввод данных.

Функция input() позволяет программе получить данные от пользователя через клавиатуру.

Базовое использование

name = input()

или с подсказкой:

```
1    name = input("Введите имя: ")
2    print("Привет,", name)
```

```
Введите имя: Ivan
Привет, Ivan
```

Функция **input()** всегда возвращает строку!

Даже если пользователь вводит число!

```
1  a = input("Введите число: ")
2  print(type(a))
```

```
Введите число: 123
<class 'str'>
```

Что такое строка?

Строка (**string**) — это тип данных в Python, который представляет собой последовательность символов, заключённых в кавычки.

Строка может содержать:

- буквы
- цифры
- пробелы
- символы пунктуации
- спецсимволы

Примеры строк:

"Hello"

"123"

"Привет, мир!"

"2025 год"

“3.14"

Все эти значения — строки, даже если они выглядят как числа.

Преобразование типов.

Когда мы получаем данные от пользователя через **input()**, они всегда приходят в виде строки.

Чтобы выполнять арифметические операции (сложение, умножение, сравнение чисел), строку нужно преобразовать в числовой тип.

Для этого используются функции:

int() — преобразует строку в целое число (если это возможно)

float() — преобразует строку в число с плавающей точкой (если это возможно)

int() — преобразование в целое число

Функция **int()** берёт строку, содержащую целое число, и превращает её в integer (целое число).

Пример:

```
1    x = int(input("Введите x: "))
2    y = int(input("Введите y: "))
3    print("Сумма:", x + y)
```

```
Введите x: 5
Введите y: 7
Сумма: 12
```

float() — преобразование в число с точкой

Числа с точкой используются для операций, где важна дробная часть: рост, температура, расстояние, деньги и т. д.

Пример:

```
price = float(input("Введите цену: "))
count = int(input("Введите количество: "))
print("Сумма: ", price * count)
```

```
Введите цену: 5.95
Введите количество: 7
Сумма:  41.65
```

Типовые ошибки.

Типовые ошибки при использовании **print()**

1. Пропущенные кавычки:

```
print(Hello)
```

```
    print(Hello)
          ^^^^^

NameError: name 'Hello' is not defined
```

2. Неправильный тип аргументов при конкатенации:

```
print("Мне " + 20 + " лет")
```

```
    print("Мне " + 20 + " лет")
          ~~~~~~~~^~~~
TypeError: can only concatenate str (not "int") to str
```

Python не может сложить строку и число.

3. Лишние или отсутствующие скобки

```
print "Hello"
```

```
    print "Hello"
    ^^^^^^^^^^^^^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

В Python обязательно нужны круглые скобки после имени вызываемой функции.

Типовые ошибки при использовании input()

1. Ожидание числа, но input() возвращает строку

```
x = input("Введите число: ")
y = x + 10
```

```
    y = x + 10
        ~~^~~~
TypeError: can only concatenate str (not "int") to str
```

Правильно:

```
x = int(input("Введите число: "))
y = x + 10
```

2. Неверный формат введённого числа:

```
x = int(input("Введите число: "))
```

```
Введите число: 100 USD
    x = int(input("Введите число: "))
        ~~~^^^^^^^^^^^^^^^^^^^^^^^^^^

ValueError: invalid literal for int() with base 10: '100 USD'
```

3. Использование input() без подсказки

Не ошибка, но плохая практика:

```
name = input()
```

Лучше:

```
name = input("Введите имя: ")
```

Итог:

Функции **print()** и **input()** — основные инструменты взаимодействия программы с пользователем.
Они позволяют:

- выводить данные в удобном формате
- принимать текстовую информацию

4. Комментарии.

Комментарий — это текст в программе, который интерпретатор игнорирует.

Комментарии нужны для пояснения кода, пометок, временного отключения строк и повышения читаемости программы.

Однострочные комментарии

Обозначаются символом **#**.
Всё, что стоит после **#**, интерпретатор не выполняет.

```
# Это комментарий
print("Hello")  # Комментарий после кода
```

Многострочные комментарии.

Специального синтаксиса для многострочных комментариев нет, но обычно используются многострочные строки """ ... """ или ''' ... '''.

```
"""
Это многострочный комментарий.
Так можно пояснять большие блоки кода.
"""
print("Программа работает")
```

Для чего нужны комментарии?

- Объяснение сложных частей кода
- Описание назначения функции, переменной или блока
- Временное отключение строки кода
- Пометки "TODO", "FIXME" для доработки

Рекомендации по использованию:

- Комментарии должны объяснять зачем и что делает код, когда код неочевиден или сложен.
- Писать кратко и по делу
- Не комментировать очевидные вещи
- Обновлять комментарии после изменения кода

Контрольные вопросы:

- Для чего используется язык Python и в каких сферах он применяется?
- В чем отличие интерпретируемого языка от компилируемого?
- Как проверить установленную версию Python в командной строке?
- Что такое интерактивный режим (REPL) и чем он отличается от запуска .py файла?
- Для чего используется функция print()? Какие у неё основные параметры?
- Как работает функция input() и какой тип данных она возвращает?
- Как создать и выполнить простую программу на Python?

Домашнее задание:

https://ru.hexlet.io/programs/python-basics-free

Материалы лекций:

https://github.com/ShViktor72/education2025

ПМ2_Прикладное_программирование

Обратная связь:

colledge20education23@gmail.com