ЦЕНТР ДОПОЛНИТЕЛЬНОГО ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Программирование на языке Python

Основные принципы работы и элементы программирования на Python

Язык программирования Python.

- **Python** – мощный и простой в использовании кросс-платформенный язык (UNIX, Linux, Windows, Mac OS, Android, iOS) общего назначения с открытым исходным кодом, поддержкой объектно-ориентированного, структурного и функционального программирования.
- Разработчик Гвидо ван Россум (Guido van Rossum).
- Официальный сайт: https://www.python.org/

- Этапы развития языка программирования Python:

- В феврале 1991 исходный код языка был опубликован на alt.sources. Язык придерживался объектно-ориентированного подхода, мог работать с классами, наследованием, функциями, обработкой исключений и всеми основными структурами данных.
- В 2000 году вышла вторая версия Python. Разработчики добавили много важных инструментов, включая поддержку Unicode и сборщик мусора.
- 3 декабря 2008 вышла третья версия Python, которая является основной до сих пор.

Версии и реализации Python.

- Одновременно существуют две линии версий реализации Python: линия версий 2 и линия версий 3.
- Параллельность связана с тем, что при переходе от версии 2 к версии 3 произошли существенные изменения синтаксиса языка, нарушившие совместимость "снизу вверх".
- В большинстве операционных систем обе версии Python могут быть инсталлированы одновременно, а то, какая версия интерпретатора будет использоваться зависит от формы запуска (команды).
- Документация версий на официальном сайте: https://www.python.org/doc/

Запуск Python. Режимы работы.

- **Python** – интерпретируемый язык.

Python может работать в двух режимах:

- Интерактивный – непосредственная интерпретация строк кода, вводимых с клавиатуры;
- Пакетный – выполнение файлов с исходным кодом.

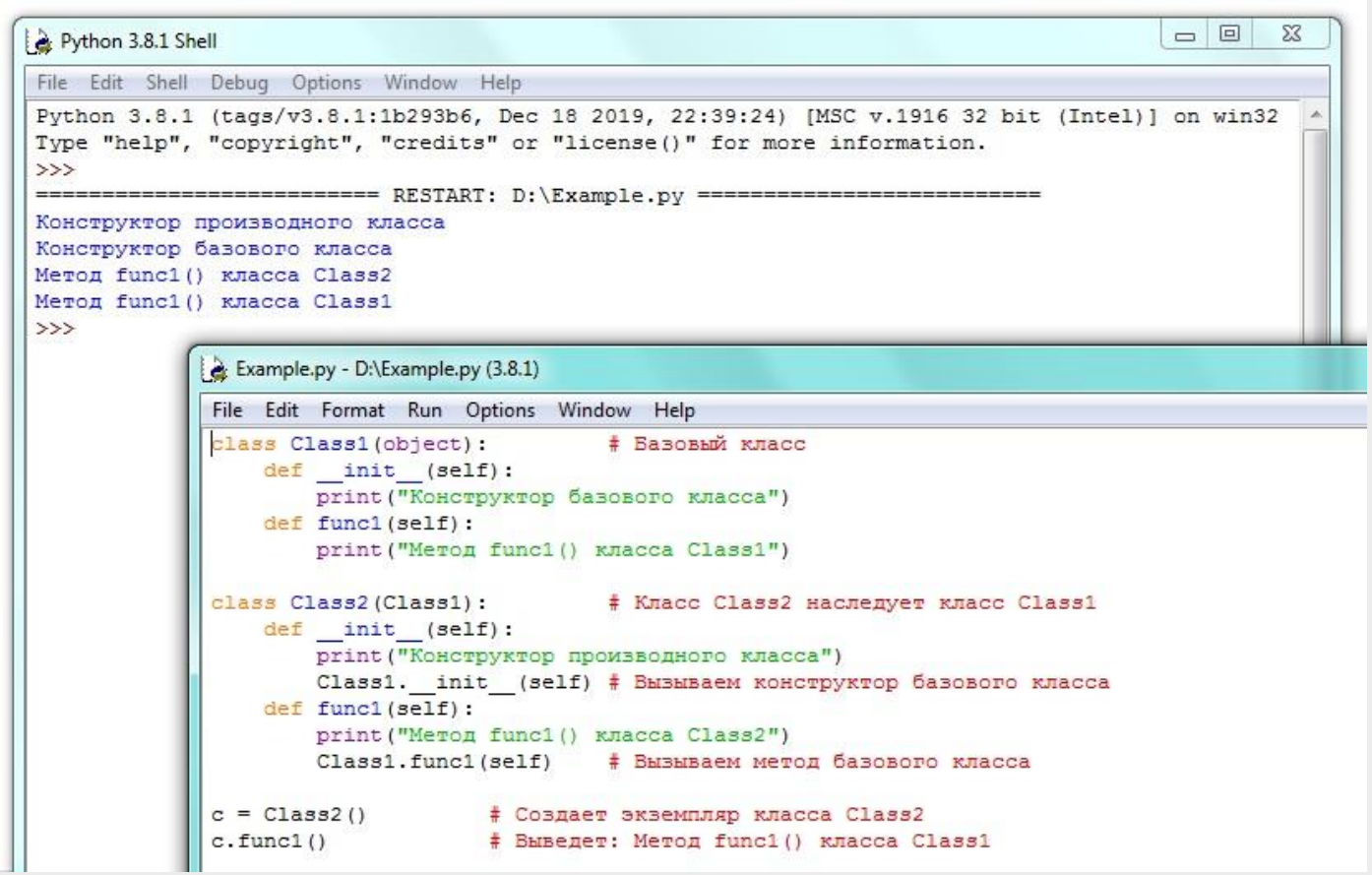
- Для запуска Python в **интерактивном режиме** необходимо набрать в командной строке имя исполняемого файла интерпретатора (**python** или **python3**) или запустить IDLE (интегрированную среду разработки).
- Для запуска Python в **пакетном режиме** необходимо ввести в командной строке имя интерпретатора и имя файла с программой на языке Python через пробел.

Пример.

```
python test.py
```

Интегрированная среда разработки.

- В комплекте вместе с интерпретатором Python поставляется *IDLE (интегрированная среда разработки)* – *интерпретатор в интерактивном* режиме с текстовым редактором и расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка и т.п.).



Комментарии.

- Комментарии предназначены для вставки пояснений в текст программы (нужны программисту, а не интерпретатору Python).
- Не обрабатываются (игнорируются) интерпретатором.

- В языке Python существует только однострочный комментарий, начинающийся с символа #, продолжающийся до конца строки:

```
# Это комментарий
```

- Однострочный комментарий может начинаться не только с начала строки, но и располагаться после команды. Например, комментарий расположен после инструкции вывода сообщения:

```
print("Привет, мир!") # Выводим надписи на экран
```

- Если символ комментария разместить перед инструкцией, то она выполнена не будет:

```
# print("Привет, мир!") Эта инструкция выполнена не будет
```

- Если символ # расположен внутри кавычек или апострофов, то он не является символом комментария:

```
print("# - это НЕ комментарий")
```

Переменные.

Переменные

Средство хранения данных во время работы программы

A + B

C – 6

a * 5 – f / 8

Идентификатор

Имя переменной (объекта)

Значение

Значение, соответствующее типу данных

Место в памяти

Участок памяти, где хранится значение

Адрес

Целочисленное значение

Именование переменных.

- Каждая переменная должна иметь уникальное имя (идентификатор), состоящее из латинских букв, цифр и знаков подчеркивания, причем имя переменной не может начинаться с цифры.
- Идентификатор – это последовательность букв, цифр и символов подчеркивания, которые начинаются с буквы или символа подчеркивания.

  Правильные имена переменных: **`x,y1,strName,str_name`**.

  Неправильные имена переменных: **`1y,имяПеременной`**.

- Следует избегать указания символа подчеркивания в начале имени, поскольку идентификаторам с таким символом определено специальное назначение.
- Имена, включающие по два символа подчеркивания – в начале и в конце, для интерпретатора имеют особый смысл.
- В качестве имени переменной нельзя использовать ключевые слова.

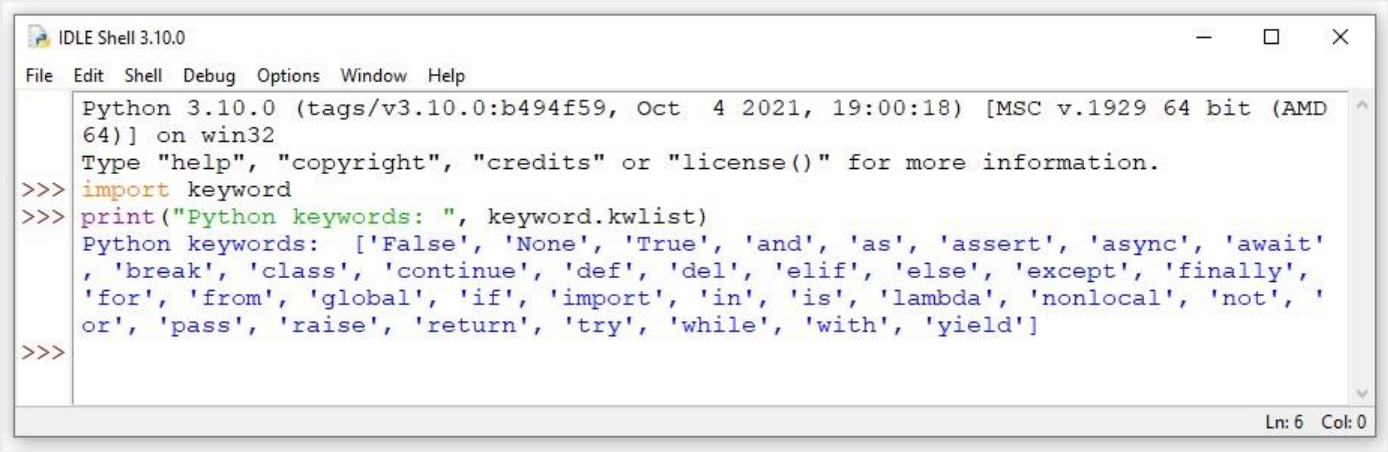
Ключевые слова.

1) False - ложь.
2) True - правда.
3) None - "пустой" объект.
4) and - логическое И.
5) with - менеджер контекста.
6) as - менеджер контекста.
7) assert условие - возбуждает исключение, если условие ложно.
8) break - выход из цикла.
9) class - пользовательский тип, состоящий из методов и атрибутов.
10) continue - переход на следующую итерацию цикла.
11) def - определение функции.
12) del - удаление объекта.
13) elif - в противном случае, если.
14) else - иначе.
15) except - перехватить исключение.
16) finally - вкупе с инструкцией try, выполняет инструкции независимо от того, было ли исключение или нет.
17) for - цикл for.
18) from - импорт нескольких функций из модуля.
19) global - делает значение переменной, присвоенное ей внутри функции, доступным и за пределами функции.
20) if - если.
21) import - импорт модуля.
22) in - проверка на вхождение.
23) is - ссылаются ли 2 объекта на одно и то же место в памяти.
24) lambda - определение анонимной функции.
25) nonlocal - делает значение переменной, присвоенное ей внутри функции, доступным в объемлющей инструкции.
26) not - логическое НЕ.
27) or - логическое ИЛИ.
28) pass - ничего не делающая конструкция.
29) raise - возбудить исключение.
30) return - вернуть результат.
31) try - выполнить инструкции, перехватывая исключения.
32) while - цикл while.
33) yield - определение функции-генератора.
34) async – асинхронный метод.
35) await - указывает, что при выполнении следующего за ним выражения возможно переключение с текущей программы на другую или на основной поток выполнения.

Ключевые слова. Модуль keyword.

- keyword.kwlist – список всех доступных ключевых слов.
- keyword.iskeyword(строка) – проверка, является ли строка ключевым словом.

```
>>> import keyword
>>> print("Python keywords: ", keyword.kwlist)
```



- Проверить является или нет идентификатор ключевым словом можно так:

```
>>> keyword.iskeyword("try")
True
>>> keyword.iskeyword("b")
False
```

Встроенные идентификаторы. Модуль builtins.

- Помимо ключевых слов, следует избегать совпадений со встроенными идентификаторами.
- В отличие от ключевых слов, встроенные идентификаторы можно переопределять, но дальнейший результат может неожиданным.

- Получение списка встроенных идентификаторов:

```
>>> import builtins
>>> dir(builtins)
```

Понятие типизации.

Python – язык с неявной сильной динамической типизацией.

- Неявная типизация – означает, что при объявлении переменной вам не нужно указывать её тип, при явной типизации – это делать необходимо.
- Динамическая типизация – тип переменной определяется непосредственно при выполнении программы.
- Статическая типизация – на этапе компиляции программы.

- **Пример.**

  C/C++/C#:

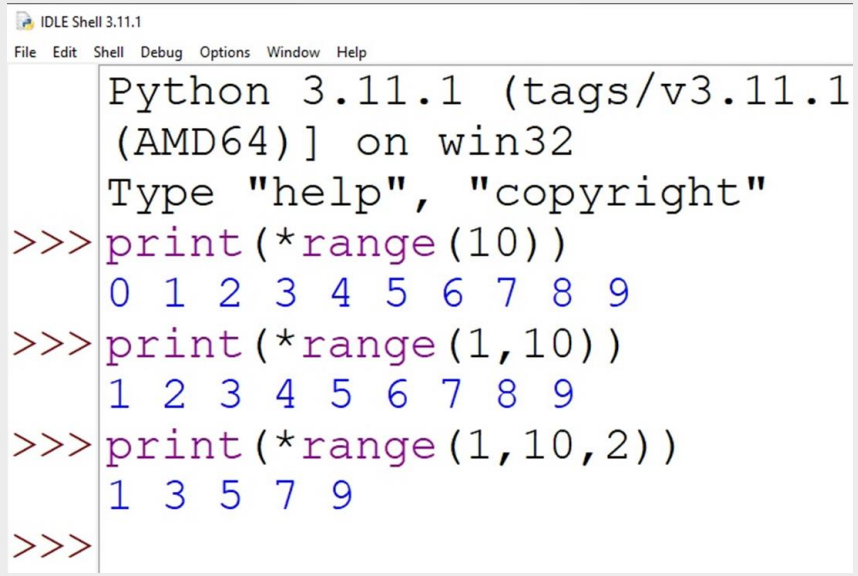
```
int a = 1;
```

  Python:

```
a = 1
```

Типы данных в Python.

- В Python типы данных можно разделить на встроенные в интерпретатор (built-in) и не встроенные, которые можно использовать при импортировании соответствующих модулей.
- К **основным встроенным типам** относятся:
  1. **None** (неопределенное значение переменной)
  2. **Логические переменные** (Boolean Type)
  3. **Числа** (Numeric Type)
     - int – целое число
     - float – число с плавающей точкой
     - complex – комплексное число
  4. **Списки** (Sequence Type)
     - list – список [1, 2, 3], [1, 2.5, ‘A’, “Python”]
     - tuple – кортеж (1, 2, 3), (1, 2.5, ‘A’, “Python”)
     - range – диапазон range(10), range(1,10), range(1,10,2)



Встроенные типы данных в Python.

5. **Строки** (Text Sequence Type, str)
   str – строка "Строка"

6. **Бинарные списки** (Binary Sequence Types)
   - ❑ bytes – байты bytes([225, 174, 92])
   - ❑ bytearray – массивы байт bytearray([225, 174, 60])

7. **Множества** (Set Types)
   - ❑ set – множество {"a", "b", "c"}
   - ❑ frozenset – неизменяемое множество
     frozenset(["a", "b", "c"])

8. **Словари** (Mapping Types)
   dict – словарь {"x": 5, "y": 20}

Изменяемые и неизменяемые типы данных.

К **неизменяемым** (immutable) типам относятся:

- целые числа (int),
- числа с плавающей точкой (float),
- комплексные числа (complex),
- логические переменные (bool),
- кортежи (tuple),
- строки (str),
- неизменяемые множества (frozen set),
- байты (bytes).

К **изменяемым** (mutable) типам относятся:

- ✓ списки (list),
- ✓ множества (set),
- ✓ словари (dict),
- ✓ массивы байт (bytearray).

Удаление переменных.

- Удалить переменную можно с помощью инструкции del:

```
del <Переменная1>[, ..., <ПеременнаяN>]
```

- Пример удаления одной переменной:

```
>>> x = 10
>>> print(x)
10
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

- Пример удаления нескольких переменных:

```
>>> a, b = 10, 20
>>> print(a,b)
10 20
>>> del a,b
>>> print(a,b)
```

Операторы в Python.

- **Операторы присваивания** предназначены для сохранения значения в переменной.

- В языке Python доступны следующие операторы присваивания:
  - ✓ **=** присваивает переменной значение
  - ✓ **:=** оператор-морж(walrus). Используется начиная с версий Python 3.8.x в «выражениях присваивания»)
  - ✓ **+=** увеличивает значение переменной на указанную величину.
    Для последовательностей оператор += производит конкатенацию
  - ✓ **-=** уменьшает значение переменной на указанную величину
  - ✓ ***=** умножает значение переменной на указанную величину
    Для последовательностей оператор *= производит повторение
  - ✓ **/=** делит значение переменной на указанную величину
  - ✓ **//=** деление с округлением вниз и присваиванием
  - ✓ **%=** деление по модулю и присваивание
  - ✓ ****=** возведение в степень и присваивание

Операторы в Python.

- Операторы позволяют произвести с данными определенные действия.
- **Математические операторы** позволяют производить операции над числами:
  - ✓ **+** сложение,
  - ✓ **-** вычитание,
  - ✓ ***** умножение,
  - ✓ **/** деление. Результатом деления всегда является вещественное число, даже если производится деление целых чисел.
  - ✓ **/ /** деление с округлением вниз. Вне зависимости от типа чисел остаток отбрасывается.
  - ✓ **%** остаток от деления,
  - ✓ ****** возведение в степень,
  - ✓ унарный минус (**-**) и унарный плюс (**+**).

Операторы в Python.

- **Операторы для работы с последовательностями:**

  - ✓ **+** конкатенация.
  - ✓ ***** повторение.
  - ✓ **[:]** извлечение среза (фрагмента) по указанному диапазону индексов.
  - ✓ **in** проверка на вхождение.

    Если элемент входит в последовательность, то возвращается логическое значение `True`.
  - ✓ **not in** проверка на невхождение.

    Если элемент не входит в последовательность, возвращается `True`.

Приоритет выполнения операторов.

- **Приоритет выполнения операторов** определяет последовательность вычисления выражений.

- **Операторы в порядке убывания приоритета:**

1) **-x, +x, ~x, **** — унарный минус, унарный плюс, двоичная инверсия, возведение в степень. Если унарные операторы расположены слева от оператора **, то возведение в степень имеет больший приоритет, а если справа - то меньший.
2) ***, %, /, / /** — умножение (повторение), остаток от деления, деление, деление с округлением вниз.
3) **+,-** — сложение (конкатенация), вычитание.
4) **<< , >>** — двоичные сдвиги.
5) **&** — двоичное И.
6) **^** — двоичное исключающее ИЛИ.
7) **|** — двоичное ИЛИ.
8) **=, :=, +=, -=, *=, /=, //=, %=, **=** — присваивание.

Работа с числами. Преобразования.

- Язык Python поддерживает следующие числовые типы:
  - **int** — целые числа. Размер числа ограничен лишь объемом оперативной памяти;
  - **float** — вещественные числа;
  - **complex** — комплексные числа.
- Операции над числами разных типов возвращают число, имеющее более сложный тип из типов, участвующих в операции.
- Целые числа имеют самый простой тип, далее идут вещественные числа и самый сложный тип - комплексные числа.
- Таким образом, если в операции участвуют целое число и вещественное, то целое число будет автоматически преобразовано в вещественное число, а затем произведена операция над вещественными числами. Результатом этой операции будет вещественное число.

Числа с фиксированной точностью. Класс Decimal.

- Числа с фиксированной точностью – числа типа Decimal, которые при вычислениях используют фиксированное количество знаков после запятой.
- Тип Decimal – специально разработанный класс (начиная с версии Python 2.4).
- Класс Decimal реализован в модуле decimal. Чтобы использовать возможности класса Decimal нужно выполнить команду:

```
from decimal import Decimal
```

- Для создания объекта класса Decimal используется конструктор этого класса. Конструктор получает строку с числом, в котором указывается заданная точность.

- Примеры.

```
Decimal('0.25')    # фиксированная точность 2 знака после запятой
Decimal('0.002')   # фиксированная точность 3 знака после запятой
Decimal('0.00001') # фиксированная точность 5 знаков после запятой
```

Числа с фиксированной точностью. Класс Decimal.

- Когда в выражении, содержащем тип Decimal, имеются числа с разной точностью представлений точность результата автоматически устанавливается по числу с наибольшей точностью представления.

- Пример.

  При сложении трех чисел:

```
>>> c=Decimal('0.1')+Decimal('0.001')+Decimal(str(0.01))
>>> print(c)
>>> 0.111
```

  автоматически устанавливается точность 3 знака после запятой, так как конструктор

```
Decimal('0.001')
```

  определяет число 0.001 с наибольшей точностью представления.

Класс Context.
Глобальная настройка точности.

- Для управления точностью типа Decimal, необходимо изменить параметр контекста prec (от англ. precision – точность) с помощью функции `getcontext()`.

- Пример.

```
>>> a = Decimal(5)/Decimal(17) # Точность не задана
>>> print('a = ', a)
a = 0.29411764705882352941176470 59

>>> # Задание точности 6 знаков после запятой
>>> getcontext().prec=6
>>> b = Decimal(5)/Decimal(17)
>>> print('b = ', b)
b = 0.294118

>>> c = Decimal(6)/Decimal(19)
>>> print('c = ', c)
c = 0.315789
```

Работа с строками.

- Строки представляют собой последовательности символов.
- Длина строки ограничена лишь объемом оперативной памяти компьютера.
- Как и все последовательности, строки поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор **+**), повторение (оператор ******), а также проверку на вхождение (операторы **in** и **not in**).
- Строки относятся к неизменяемым типам данных. Поэтому практически все строковые методы в качестве значения возвращают новую строку *(при использовании небольших строк это не приводит к каким-либо проблемам, но при работе с большими строками можно столкнуться с проблемой нехватки памяти).*
- Можно получить символ по индексу, но изменить его будет нельзя:

```
>>> s = "Python"
>>> s[0]          # Можно получить символ по индексу
'P'
>>> s [0] = "J"   # Изменить нельзя, сообщение об ошибке
```

Вывод результатов работы программы.

- Вывести на экран результаты работы программы можно с помощью функции print(). Функция имеет следующий формат:

```
print([<Объекты>] [, sep=' '][, end='\n'][/ file=sys.stdout] [, flush=False])
```

- Функция print() преобразует объект в строку и посылает ее в стандартный поток вывода stdout.
- С помощью параметра file можно перенаправить вывод в файл.
- После вывода строки автоматически добавляется символ перевода строки:

```
print("Строка 1")
print("Строка 2")
```

Результат:

```
Строка 1
Строка 2
```

Вывод результатов работы программы.

- Если необходимо вывести результат на той же строке, то в функции print() данные указываются через запятую в первом параметре:

```
print("Строка1", "Строка2")
```

Результат:

```
Строка1 Строка2
```

Между выводимыми строками автоматически вставляется пробел.

- С помощью параметра sep можно указать другой символ. Пример. Вывод строк без пробела между ними:

```
print("Строка1", "Строка2", sep="")
```

Результат:

```
Строка1Строка2
```

Вывод результатов работы программы.

- После вывода объектов в конце добавляется символ перевода строки. Если необходимо произвести дальнейший вывод на той же строке, то в параметре end следует указать другой символ:

```
print("Строка 1", "Строка 2", end=" ")
print("Строка 3")
```

Результат:

```
Строка 1 Строка 2 Строка 3
```

- Если необходимо вывести многострочный блок текста, его следует разместить между утроенными апострофами. В этом случае текст сохранит свое форматирование:

```
print("""Строка 1
Строка 2
Строка 3""")
```

Результат – три строки:

```
Строка 1
Строка 2
Строка 3
```

Вывод результатов с помощью f-строк.

- f-строки: новый и улучшенный способ форматирования строк в Python!
- f-строки были представлены в Python 3.6 (PEP 498, август 2015 года https://www.python.org/dev/peps/pep-0498).
- f-строки – строковые литералы, которые имеют f в начале и фигурные скобки, содержащие выражения, которые будут заменены их значениями.
- Выражения оцениваются во время выполнения, а затем форматируются с использованием протокола format.
- Формат:

  `f'{value:{width}.{precision}}'`, где:

  - ✓ `value` – любое выражение, которое вычисляет число (имя переменной);
  - ✓ `width` – задает количество символов, используемых в общей сложности, но если value требуется больше места, чем указывает ширина, используется дополнительное пространство;
  - ✓ `precision` - указывает количество символов после запятой точки.

```
print(f'\nСтоимость: {summ:6.2f} руб., расстояние: {s:.0f} км.')
```

Вывод результатов работы программы.

- Для вывода результатов работы программы вместо функции print() можно использовать метод write() объекта sys.stdout:

```
import sys                          # Подключаем модуль sys
sys.stdout.write("Строка") # Выводим строку
```

- Метод write() не вставляет символ перевода строки, поэтому при необходимости следует добавить его с помощью символа \n:

```
import sys
sys.stdout.write("Строка 1\n")
sys.stdout.write("Строка 2")
```

Результат:

```
Строка 1
Строка 2
```

Ввод данных.

- Для ввода данных в Python предназначена функция input(), которая получает данные со стандартного потока ввода stdin.
- Функция имеет следующий формат:
- `[<Значение> = ] input([<Сообщение>])`
- Пример. Программа выводит сообщение и ожидает ввода, после чего выводит приветствие.

```
name = input("Введите ваше имя: ")
print("Привет,", name, "!")
input("Нажмите <Enter> для завершения...")
```

Результат:

```
Введите ваше имя: Иван
Привет, Иван !
Нажмите <Enter> для завершения...
```

Практикум.

- Разработка простых программ.
- Использование функций ввода-вывода.
- Работа с данными и строками.

Литература.

- Программируем на Python / Доусон М. – СПб.: Питер, 2021 – 416 с.

- Python. Карманный справочник / Марк Лутц, 5-е изд. – ООО “И.Д.Вильямс”, 2020. – 320с. : ил.

- Глубокое обучение на Python. / Шолле Франсуа – СПб.: Питер, 2023. – 400 с.: ил. – (Серия «Библиотека программиста»).

- Изучаем программирование на Python / Пол Бэрри ; [пер. с англ. М.А. Райтман]. – Москва : Издательство «Э», 2022. – 624 с. : ил. – (Мировой компьютерный бестселлер).

ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

do.bmstu.ru