СОДЕРЖАНИЕ

Введение в программирование

Представьте, что вы пишете список покупок на завтра. В этот момент вы работаете как программист для своего мозга: вы даёте ему инструкции, что купить. Программирование — это примерно то же самое, только ваши "инструкции" даются не человеку, а компьютеру. Программа — это набор таких инструкций, которые компьютер выполняет по порядку, чтобы достигнуть определённой цели.

Python — один из самых популярных языков программирования, потому что он лёгок в освоении и универсален. Компьютер понимает написанный вами код с помощью **интерпретатора** — это как переводчик с человеческого языка на язык компьютера. А чтобы взаимодействовать с интерпретатором, мы используем **консоль** — текстовое окно, где вводятся команды и отображаются результаты.

Вывод сообщений на экран: как сказать компьютеру: "Привет!"

Когда вы хотите "поговорить" с компьютером, первый шаг — научиться выводить сообщения на экран. Это можно сравнить с отправкой сообщения другу. Например, чтобы компьютер напечатал "Привет, мир!", используйте функцию **print()**:

```
print("Привет, мир!")
```

Для вывода текста вы можете использовать одинарные (') или двойные (") кавычки:

```
print('Программирование — это интересно!')
print("Начнём учиться вместе.")
```

Разница между ними в том, как удобнее работать с разными символами внутри строки. Если вам нужно вывести текст с кавычками, то лучше использовать противоположные кавычки:

```
print('Он сказал: "Давай учиться Python".')
```

Рекомендуется придерживаться одного стиля: используйте всегда двойные кавычки — так код будет выглядеть аккуратнее и меньше шансов допустить ошибку.

Стандарты оформления кода: PEP8

В Python существует стандарт оформления кода — **PEP8**. Это набор рекомендаций по стилю написания кода на Python, который помогает сделать его читаемым и удобным для других программистов. Рассмотрим несколько важных правил, которые будут использоваться в нашем коде:

После функции print() скобки пишутся без пробелов:

```
print("Hello")  # правильно
print ( "Hello" )  # неправильно
```

Что такое переменные?

Переменные в программировании похожи на ящики для хранения вещей. Представьте, что вы складываете в коробку яблоки. Коробка с надписью "фрукты" помогает вам запомнить, что внутри. В программировании "коробка" — это переменная, а то, что вы в неё кладёте, — это данные. Другими словами: **переменные** — это контейнеры для хранения данных.

Пример создания переменной:

```
age = 20
```

Здесь переменная age хранит значение 20, которое можно использовать позже. Важно, чтобы название переменной отображало, что именно она хранит. Например, если переменная содержит возраст, назовите её age, а не x. Это делает код понятным.

Переменные позволяют хранить и манипулировать данными в программе. При выборе названия переменной следует придерживаться следующих правил:

1) Имена должны начинаться с буквы или символа подчёркивания.

2) Они могут содержать буквы, цифры и символ подчёркивания, но не могут начинаться с цифры.

3) Имена переменных чувствительны к регистру (переменные age и Age — это разные переменные).

В Python принято писать имена переменных строчными буквами с разделением слов с помощью символа подчёркивания, как в примере:

```
student_age = 20
```

Ошибки, которые часто встречаются:

1) Использование пробелов вокруг знака =. Правильно: age = 20, неправильно: age=20.

2) Названия переменных не должны содержать пробелов, правильно: student_age, неправильно: student age.

Ввод данных с клавиатуры

Чтобы программа могла взаимодействовать с пользователем, используются данные, вводимые с клавиатуры. Это делается с помощью функции **input()**. Например, чтобы попросить ввести имя:

```
name = input()
```

Это как будто компьютер спрашивает: "Как тебя зовут?". Здесь пользователь вводит своё имя, и оно сохраняется в переменную name. Но так как всё, что вводит пользователь, воспринимается как текст (даже числа), если нужно ввести число, его надо преобразовать.

Целые числа и работа с ними

Целые числа (integer) — это числа без дробной части. Чтобы считать целое число с клавиатуры, необходимо преобразовать ввод, так как функция input() всегда возвращает строку. Это делается с помощью функции **int()**:

```
number = int(input())
```

В программировании часто нужно выполнять расчёты. Вот таблица с основными операциями:

Операция	Символ	Пример в коде	Пример из математики
Сложение	+	x = 5 + 3	5 + 3 = 8
Вычитание	-	x = 9 - 4	9 - 4 = 5
Умножение	*	x = 7 * 2	7 * 2 = 14
Деление	/	x = 8 / 2	8 / 2 = 4.0
Деление нацело	//	x = 9 // 4	9 // 4 = 2
Остаток от деления	%	x = 9 % 4	9 % 4 = 1
Возведение в степень	**	x = 2 ** 3	$2^3 = 8$

Приоритет выполнения операций в Python определяет, в каком порядке выполняются математические действия в одном выражении. Это важно для правильных вычислений. Давайте рассмотрим каждую математическую операцию и её приоритет. Приоритет операций в Python:

1) Возведение в степень **

2) Умножение, деление, деление нацело, остаток от деления *, /, //, %

3) Сложение и вычитание +, -

Допустим, у нас есть выражение:

```
result = 5 + 2 * 3 ** 2 // 4 - 1
```

Разберём его по шагам, учитывая приоритет операций:

1) Сначала возведение в степень 3 ** 2 = 9 (наивысший приоритет).

2) Затем умножение: 2 * 9 = 18.

3) После этого деление нацело: 18 // 4 = 4.

4) Теперь выполняем сложение: 5 + 4 = 9.

5) И, наконец, вычитание: 9 - 1 = 8.

Итоговый результат будет 8.

Если хотите изменить приоритет операций, можно использовать скобки. Всё, что находится внутри скобок, выполняется в первую очередь, вне зависимости от приоритета операции.

Кроме приоритета операций, в Python важно учитывать ассоциативность операций — порядок, в котором выполняются действия с одинаковым приоритетом. В большинстве случаев операции с одинаковым приоритетом выполняются слева направо (это называется левая ассоциативность). Однако есть исключения, как возведение в степень.

Возведение в степень (**) является правоассоциативной операцией, что означает, что она выполняется справа налево:

```
result = 2 ** 3 ** 2
```

Это выражение вычисляется как:

1) Сначала вычисляется возведение 3 в квадрат: 3 ** 2 = 9.

2) Затем 2 возводится в степень 9: 2 ** 9 = 512.

Итог: result = 512.

Пример с одинаковым приоритетом:

```
result = 16 // 4 * 2 % 3
```

Все операции здесь (деление нацело, умножение и остаток от деления) имеют одинаковый приоритет, поэтому они выполняются слева направо:

1) Сначала выполняем деление нацело: 16 // 4 = 4.

2) Затем умножаем результат: 4 * 2 = 8.

3) Наконец, находим остаток от деления: 8 % 3 = 2.

Итог: result = 2.

Следите за тем, чтобы в арифметических выражениях пробелы вокруг операторов сохраняли читаемость:

```
x = 5 + 3  # правильно
x=5+3  # неправильно
```

Разбиение числа на разряды и нахождение суммы цифр

Представьте себе четырёхзначное число, например, 1234. Как его разбить на цифры? Мы можем использовать целочисленное деление и остаток от деления. Рассмотрим следующий пример:

```python
number = int(input())  # Вводим четырёхзначное число

digit1 = number % 10  # Последняя цифра
digit2 = (number // 10) % 10  # Вторая цифра
digit3 = (number // 100) % 10  # Третья цифра
digit4 = number // 1000  # Первая цифра

# Суммируем цифры
sum_of_digits = digit1 + digit2 + digit3 + digit4

# Выводим результат
print(sum_of_digits)  # Например, если число 1234, сумма будет 10
```

Функция print() и её возможности

print() может вывести неограниченное количество аргументов, разделённых пробелами:

```python
print(1, 2, 3, 4)  # Выводит: 1 2 3 4
```

Функция поддерживает два полезных именованных аргумента:

1) sep — это разделитель, который используется между аргументами. По умолчанию это пробел:

```python
print(1, 2, 3, sep='-')  # Вывод: 1-2-3
```

2) end — определяет, чем завершить вывод. По умолчанию — это новая строка:

```python
print(1, 2, 3, end='; ')
print(4, 5)  # Вывод: 1 2 3; 4 5
```

Важно помнить, что sep и end влияют только на тот print(), в котором они используются.

При изменении аргументов функции, таких как sep и end, не ставятся пробелы вокруг знака присваивания:

```
print(1, 2, 3, sep='-')  # правильно
print(1, 2, 3, sep = '-')  # неправильно
```

Форматированные строки (f-строки)

Для удобства вывода переменных используйте f-строки. Представьте, что вы отправляете письмо и вставляете в него конкретные данные. F-строки работают по такому же принципу:

```
name = "Иван"
age = 25
print(f"Меня зовут {name}, мне {age} лет.")
```

В результате на экран выводится: "Меня зовут Иван, мне 25 лет."

Здесь в фигурных скобках находятся переменные name и age. Python автоматически подставляет их значения в строку.

Фигурные скобки {} внутри f-строк используются как "место" для вставки значений переменных или выражений. Всё, что написано внутри этих скобок, будет вычислено и вставлено в итоговую строку.

В фигурных скобках можно размещать не только переменные, но и любые выражения: арифметические операции, вызовы функций, и даже форматирование. Пример:

```
x = 10
y = 5
print(f"{x} + {y} = {x + y}")  # Вывод: 10 + 5 = 15
```

Здесь в скобках {x + y} Python сначала выполнит сложение x и y, а затем подставит результат в строку.

Ещё примеры использования f-строк:

1) Вывод переменных:

```
price = 100
quantity = 5
print(f"Цена: {price}, количество: {quantity}, итог: {price * quantity}")
# Вывод: Цена: 100, количество: 5, итог: 500
```

2) Выполнение операций прямо в строке:

```
number = 9
print(f"Квадрат числа {number} = {number ** 2}")  # Вывод: Квадрат числа 9 = 81
```