BACKEND - РАЗРАБОТКА НА ЯЗЫКЕ PYTHON

РЭУ ИМ. Г.В. ПЛЕХАНОВА МПТ

https://ucmpt.ru/

https://do.rea.ru/

PYTHON

- Python - это высокоуровневый, интерпретируемый язык программирования общего назначения. Его философия дизайна подчеркивает удобочитаемость кода с использованием значительных отступов.
- Python динамически типизируется и собирает мусор. Он поддерживает несколько парадигм программирования, включая структурированное (особенно процедурное), объектно-ориентированное и функциональное программирование. Его часто описывают как язык с "включенными батарейками" из-за его обширной стандартной библиотеки.

ИСТОРИЯ СОЗДАНИЯ PYTHON

- Гвидо ван Россум задумал Python в 1980-х годах, а приступил к его созданию в декабре 1989 года в центре математики и информатики в Нидерландах. Язык Python был задуман как потомок языка программирования ABC, способный к обработке исключений и взаимодействию с операционной системой Амёба. Ван Россум является основным автором Python и продолжал выполнять центральную роль в принятии решений относительно развития языка вплоть до 12 июля 2018 года.
- Python 1.0 появился в январе 1994 года. Основными новыми возможностями, включёнными в этот релиз, были средства функционального программирования.
- В версии Python 2.0 добавлена система сборки мусора с поддержкой циклических ссылок. Главным нововведением в Python 2.2 было объединение базовых типов Python и классов, создаваемых пользователем, в одной иерархии. Это сделало Python полностью объектно-ориентированным языком.
- Python 3.0 разрабатывался с целью устранения фундаментальных изъянов в языке. Эти изменения не могли быть сделаны при условии сохранения полной обратной совместимости с 2.x версией, поэтому потребовалось изменение главного номера версии.

ФИЛОСОФИЯ PYTHON

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.

ФИЛОСОФИЯ PYTHON

- Ошибки никогда не должны замалчиваться.
- Если они не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один и, желательно, только один очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить, то это плохая идея.
- Если реализацию легко объяснить, то идея, возможно хороша.
- Пространства имён - отличная штука! Будем делать их больше!

ОСНОВНЫЕ ОСОБЕННОСТИ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

- Скриптовый язык. Код программ определяется в виде скриптов.
- Поддержка самых различных парадигм программирования, в том числе объектно-ориентированной и функциональной парадигм.
- Интерпретация программ. Для работы со скриптами необходим интерпретатор, который запускает и выполняет скрипт.
- Выполнение программы на Python выглядит следующим образом. Сначала мы пишем в текстовом редакторе скрипт с набором выражений на данном языке программирования. Передаем этот скрипт на выполнение интерпретатору. Интерпретатор транслирует код в промежуточный байткод, а затем виртуальная машина переводит полученный байткод в набор инструкций, которые выполняются операционной системой.

ОСНОВНЫЕ ОСОБЕННОСТИ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

- Здесь стоит отметить, что хотя формально трансляция интерпретатором исходного кода в байткод и перевод байт кода виртуальной машиной в набор машинных команд представляют два разных процесса, но фактически они объединены в самом интерпретаторе.
- Портативность и платформонезависимость. Не имеет значения, какая у нас операционная система - Windows, MacOS, Linux, нам достаточно написать скрипт, который будет запускаться на всех этих ОС при наличии интерпретатора
- Автоматическое управление памяти
- Динамическая типизация

ПРИМЕНЕНИЕ PYTHON

- **Но можно выделить 3 самых популярных направления применения Python:**
  - **Веб-разработка;**
  - **Машинное обучение, анализ данных и визуализация;**
  - **Автоматизация процессов.**

ПЕРЕМЕННЫЕ

- **Переменная – поименованная, либо адресуемая иным способом область памяти, адрес которой можно использовать для осуществления доступа к данным и изменять значение в ходе выполнения программы.**
- **Переменные предназначены для хранения данных. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить:**
- **False, await, else, import, pass, None, break, except, in, raise, True, class, finally, is, return, and, continue, for, lambda, try, as, def, from, nonlocal, while, assert, del, global, not, with, async, elif, if, or, yield.**

КАК РАБОТАТЬ С ПЕРЕМЕННЫМИ В PYTHON

- **Каждый элемент данных в Python является объектом определенного типа или класса. Когда, в процессе выполнения программного кода, появляется новое значение, интерпретатор выделяет для него область памяти – то есть создаёт объект определенного типа (число, строка и т.д.). После этого Python записывает в свой внутренний список адрес этого объекта.**

ВИДЫ ТИПЫ ДАННЫХ В PYTHON

- **В Python есть несколько стандартных типов данных:**
  - **Numbers (числа)**
  - **Strings (строки)**
  - **Lists (списки)**
  - **Dictionaries (словари)**
  - **Tuples (кортежи)**
  - **Sets (множества)**
  - **Boolean (логический тип данных)**

ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННЫХ В PYTHON

- Чтобы создать новую переменную в Python, ее не нужно заранее инициализировать – достаточно придумать ей имя и присвоить значение через оператор =.

Синтаксис:

```
a = 123
print(a)
```

Вывод: 123

```
print(type(a))
```

Вывод: <class 'int'>

```
print(id(a))
```

Вывод: 1827204944

КОНСОЛЬНЫЙ ВВОД И ВЫВОД

- Для вывода информации используется функция print(), при вызове этой функции ей в скобках передается выводимое значение. Можно передать строку, чтобы вывести какое-либо сообщение. Можно вложить переменную, в таком случае будет выведено его значение.
- Пример:

```
print(“Hello World”)
```

Вывод: Hello World

```
a = 18
print(a)
```

Вывод: 18

КОНСОЛЬНЫЙ ВВОД И ВЫВОД

- Наряду с выводом на консоль мы можем получать ввод пользователя с консоли, получать вводимые данные. Для этого в Python определена функция input(). В эту функцию передается приглашение к вводу. А результат ввода мы можем сохранить в переменную.
- Пример:

```
name = input("Введите любое слово: ")
print("Слово: ", name)
```

Вывод: Введите любое слово: яблоко

Слово: яблоко

ВЫРАЖЕНИЯ В PYTHON

- **Выражение – один из типов инструкции, который содержит логическую последовательность чисел, строк, объектов и операторов python. Значение и переменная являются выражениями сами по себе. С помощью выражений можно выполнять такие операции, как сложение, вычитание, конкатенация и другие.**

ОПЕРАТОРЫ В PYTHON

- Арифметические операторы
- Операторы сравнения
- Операторы присваивания
- Логические операторы
- Операторы принадлежности
- Операторы тождественности
- Битовые операторы

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- **Сложение (+)**
- **Складывает значение по обе стороны оператора.**
- **Пример:**

3+4

Вывод: 7

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- **Вычитание (-)**
- **Вычитает значение правой стороны из значения в левой.**
- **Пример:**

3-4

Вывод: -1

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- **Умножение (*)**
- **Перемножает значения с обеих сторон оператора.**
- **Пример:**

3*4

Вывод: 12

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- **Деление (/)**
- **Делит значение левой стороны на значение правой. Тип данных результата деления – число с плавающей точкой.**
- **Пример:**

3/4

Вывод: 0.75

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- Возведение в степень (**)
- Возводит первое число в степень второго.
- Пример:

3**4

Вывод: 81

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- **Деление без остатка (//)**
- **Выполняет деление и возвращает целочисленное значение частного, убирая цифры после десятичной точки.**
- **Пример:**

4//3

Вывод: 1

10//3

Вывод: 3

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

- Деление по модулю (остаток от деления) (%)
- Выполняет деление и возвращает значение остатка.
- Пример:

3%4
Вывод: 3
4%3
Вывод: 1
10%3
Вывод: 1
10.5%3
Вывод: 1.5

ОПЕРАТОРЫ СРАВНЕНИЯ

- Операторы сравнения в Python проводят сравнение операндов. Они сообщают, является ли один из них больше второго, меньше, равным или и то и то.

- Меньше (<)
- Этот оператор проверяет, является ли значение слева меньше, чем правое.
- Пример:

4<3

Вывод: False

ОПЕРАТОРЫ СРАВНЕНИЯ

- **Больше (>)**
- **Проверяет, является ли значение слева больше правого.**

- **Пример:**

4>3

Вывод: True

ОПЕРАТОРЫ СРАВНЕНИЯ

- **Меньше или равно (<=)**
- **Проверяет, является ли левая часть меньше или равной правой.**
- **Пример:**

7<=7

Вывод: True

ОПЕРАТОРЫ СРАВНЕНИЯ

- **Больше или равно (>=)**
- **Проверяет, является ли левая часть больше или равной правой.**
- **Пример:**

0>=0

Вывод: True

ОПЕРАТОРЫ СРАВНЕНИЯ

- Равно (==)
- Этот оператор проверяет, равно ли значение слева правому. 1 равна булевому True, а 2 (двойка) – нет. 0 равен False.
- Пример:

3==3.0

Вывод: True

1==True

Вывод: True

7==True

Вывод: False

ОПЕРАТОРЫ СРАВНЕНИЯ

- Не равно (!=)
- Проверяет, не равно ли значение слева правому. Оператор <> выполняет ту же задачу, но его убрали в Python 3.

- Когда условие выполнено, возвращается True. В противном случае – False. Это возвращаемое значение можно использовать в последующих инструкциях и выражениях.
- Пример:

```
1!=1.0
Вывод: False
1==True  # Это вызывает SyntaxError
Вывод: SyntaxError
```

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Оператор присваивания присваивает значение переменной. Он может манипулировать значением до присваивания. Есть 8 операторов присваивания: 1 простой и 7 с использованием арифметических операторов.**

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- Присваивание (=)
- Присваивает значение справа левой части. Стоит обратить внимание, что == используется для сравнения, а = – для присваивания.
- Пример:

```
a = 7
print(a)
```

Вывод: 7

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- Сложение и присваивание (+=)
- Суммирует значение обеих сторон и присваивает его выражению слева. a += 10 – это то же самое, что и a = a + 10.

- То же касается и все остальных операторов присваивания.
- Пример:

```
a += 2
print(a)
```

Вывод: 9

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Вычитание и присваивание (-=)**
- **Вычитает значение справа из левого и присваивает его выражению слева.**
- **Пример:**

```
a -= 2
print(a)
```

Вывод: 7

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Деление и присваивание (/=)**
- **Делит значение слева на правое. Затем присваивает его выражению слева.**
- **Пример:**

```
a /= 7
print(a)
```

Вывод: 1.0

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Умножение и присваивание (*=)**
- **Перемножает значения обеих сторон. Затем присваивает правое левому.**
- **Пример:**

```
a *= 8
print(a)
```

Вывод: 8.0

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Деление по модулю и присваивание (%=)**
- **Выполняет деление по модулю для обеих частей. Результат присваивает левой части.**
- **Пример:**

```
a %= 3
print(a)
```

Вывод: 2.0

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- **Возведение в степень и присваивание (**=)**
- **Выполняет возведение левой части в степень значения правой части. Затем присваивает значение левой части.**
- **Пример:**

```
a **= 5
print(a)
```

Вывод: 32.0

ОПЕРАТОРЫ ПРИСВАИВАНИЯ

- Деление с остатком и присваивание (//=)
- Выполняет деление с остатком и присваивает результат левой части.
- Пример:

```
a //= 3
print(a)
```

Вывод: 10.0

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ В PYTHON

- Это союзы, которые позволяют объединять по несколько условий. В Python есть всего три оператора: and (и), or (или) и not (не).

- И (and)
- Если условия с двух сторон оператора and истинны, тогда все выражение целиком считается истинным.
- Пример:

7 > 7 and 2 > -1

Вывод: False

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ В PYTHON

- Или (or)
- Если условия с одной из сторон оператора or истинно, тогда все выражение целиком считается истинным.
- Пример:

```
7 > 7 or 2 > -1
```

Вывод: True

ЛОГИЧЕСКИЕ ОПЕРАТОРЫ В PYTHON

- **Не (not)**
- **Этот оператор инвертирует булевые значения выражения. True превращается в False и наоборот. В примере внизу булево значение 0 – False. Поэтому оно превращается в True.**
- **Пример:**

```
a = not(0)
print(a)
```

Вывод: True

В PYTHON СУЩЕСТВУЮТ СЛЕДУЮЩИЕ УСЛОВНЫЕ КОНСТРУКЦИИ:

- if
- If / else
- if / elif / else
- вложенные if конструкции.

УСЛОВНАЯ КОНСТРУКЦИЯ IF В PYTHON:

- Команда if в Python работает по той же схеме, что и в других языках программирования. Она содержит в себе логическое условие, и если это условие истинно (равно True) - выполнится блок кода, записанный внутри команды if. Если же логическое условие ложно (равно False), то блок кода записанный внутри команды if пропускается, а выполнение кода переходит на следующую после блока if строчку кода.

- Например:

```
# Происходит сравнение переменной num с нулем
if num > 0:
    print "Число больше нуля" # Если переменная больше нуля, то печатается строка
print "Строка после блока if" # Данная строка печатается в любом случае, поскольку она находится вне блока if
```

КОНСТРУКЦИЯ IF...ELSE В PYTHON:

- В конструкцию if может быть добавлена команда else. Она содержит блок кода, который выполняется, если условие в команде if ложно.
- Команда else является опциональной, в каждой if - конструкции может быть только одна команда else.
- Например:

```
if num > 0:
    print "Число больше нуля" # если переменная num больше нуля то выполняется этот блок кода
else:
    print "Число меньше или равно нулю" # иначе выполнится этот блок кода
```

КОМАНДА ELIF В PYTHON:

- Команда elif позволяет проверить истинность нескольких выражений и в зависимости от результата проверки, выполнить нужный блок кода.

- Как и команда else, команда elif являктся опциональной, однако, в отличие от команды else, у одной if-конструкции может существовать произвольное количество команд elif.

- Например:

```
# Производится последовательное сравнение переменной num.
# Если num больше ста выполняется код в строке 4 и выполнение переходит на строку 13, иначе выполняется проверка в строке 6
if num > 100:
        print "Число больше ста"
# Если num больше пятидесяти - выполняется код в строке 7 и выполнение переходит на строку 13, иначе выполняется проверка в строке 8 и т.д.
elif num > 50:
        print "Число больше пятидесяти"
elif num > 25:
        print "Число больше двадцати пяти"
# Если результат всех проверок оказался ложным выполняется блок в строке 11, после чего переходим на строку 13
else:
        print "Число меньше двадцати пяти"
print "Финальная строка"
```

ВЛОЖЕННЫЕ УСЛОВНЫЕ КОНСТРУКЦИИ В PYTHON:

- В процессе разработки может возникнуть ситуация, в которой после одной истинной проверки следует сделать еще несколько. В таком случае необходимо использовать вложенные условные конструкции. То есть одну if...elif...else конструкцию внутри другой.

- Например:

```
if num > 100:
        if num < 150:
                print "Число больше ста, но меньше ста пятидесяти"
        elif num < 200:
                print "Число больше ста, но меньше двухсот"
elif num > 50:
        if num < 90:
                print "Число больше пятидесяти, но меньше девяноста«
        else:
                print "Число больше пятидесяти и больше девяноста"
else:
        print "Число меньше пятидесяти"
```

- Логика выполнения вложенных условных конструкций та же, что и у обычных. Главное не запутаться с отступами и порядком выполнения сравнений.

МАССИВЫ В PYTHON

- Массив - это структура данных, в которой хранятся значения одного типа.
- Список – это тип данных, который хранит набор или последовательность элементов. При этом в списке может храниться данные различных типов. Это основное различие между массивами и списками. В Python нет таковых массивов, вместо их использует списки.
- Диапазон – это неизменяемый последовательный набор чисел.
- Множества – это последовательность значений, который хранит только уникальные элементы.
- Кортеж – это последовательность элементов, который является неизменяемым типом.
- Словари – это коллекция элементов, где каждый элемент имеет уникальный ключ и ассоциированое с ним некоторое значение (ключ-значение, как в JSON).

ОБЪЯВЛЕНИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ PYTHON

- Список

numbers = [1,2,3,4,5]

words = [“Python”,”integer”,”car”,”Apple”]

- Диапазон

range(5)

range(1, 5)

range(2, 10, 2)

ОБЪЯВЛЕНИЯ ПОСЛЕДОВАТЕЛЬНОСТЕЙ PYTHON

- Множества

```
users = {"Tom", "Bob", "Alice", "Tom"}
```

- Кортеж

```
tom = ("Tom", 23)
```

- Словари

```
users = {1: "Tom", 2: "Bob", 3: "Bill"}
```

ОПЕРАТОРЫ ПРИНАДЛЕЖНОСТИ

- Эти операторы проверяют, является ли значение частью последовательности. Последовательность может быть списком, строкой или кортежем. Есть всего два таких оператора: in и not in.

- В (in)
- Проверяет, является ли значение членом последовательности. В этом примере видно, что строки fox нет в списке питомцев. Но cat – есть, поэтому она возвращает True. Также строка me является подстрокой disappointment. Поэтому она вернет True.
- Пример:

```
pets=['dog','cat', 'ferret']
'fox' in pets
```

Вывод: False

ОПЕРАТОРЫ ПРИНАДЛЕЖНОСТИ

- Нет в (not in)
- Этот оператор проверяет, НЕ является ли значение членом последовательности.
- Пример:

'pot' not in 'disappointment'

Вывод: True

ОПЕРАТОРЫ ТОЖДЕСТВЕННОСТИ

- Эти операторы проверяют, являются ли операнды одинаковыми (занимают ли они одну и ту же позицию в памяти).

- Это (is)
- Если операнды тождественны, то вернется True. В противном случае – False. Здесь 2 не является 20, поэтому вернется False. Но '2' – это то же самое, что и "2". Разные кавычки не меняют сами объекты, поэтому вернется True.
- Пример:

2 is 20

Вывод: False

'2' is "2"

Вывод: True

ОПЕРАТОРЫ ТОЖДЕСТВЕННОСТИ

- Это не (is not)
- 2 – это число, а '2' – строка. Поэтому вернется True.
- Пример:

2 is not '2'

Вывод: True

СТРОКИ В PYTHON

- Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.
- Пример

```
message = “Hello World”

print(message)
```

Вывод: Hello World

ЦИКЛЫ

- **Цикл - разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.**
- **В Python существует 2 типа циклов:**
  - **while;**
  - **for;**

ЦИКЛ WHILE

- Цикл while (с англ. - "пока") позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл while используется, когда невозможно определить точное значение количества проходов исполнения цикла.
- Пример использования:

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

ЦИКЛ FOR

- Цикл for, также называемый циклом с параметром, в языке Питон богат возможностями. В цикле for указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном.
- Пример использования:

```
j = 1
for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
    print('#', i, ' color of rainbow is ', color, sep = '')
    j += 1
```