Урок1

Работа в команде

Что такое спринт?

Спринты — это регулярные ограниченные промежутки времени, в течении которых команда выполняет заданный объем работы в рамках большого проекта. Спринт позволяет сделать работу над проектом более гибкой, прозрачной и удобной как для заказчика, так и для команды разработчиков.

Сколько обычно длится спринт?

Длина и объем задач спринта определяются заранее и не могут быть изменены в процессе. Как правило, длина спринта равна четырем неделям. Но бывают варианты и короче — спринты длиной в одну-две недели. Сразу после окончания одного спринта начинается следующий.

Что такое бэклог?

Бэклог продукта — это перечень задач, которые необходимо выполнить в ходе работы над проектом, и список функций, которые хотят получить пользователи и заинтересованные лица. В него входят как уже запланированные шаги, так и пожелания заинтересованных лиц по улучшению продукта.

Приведите пример проверки продуктовой гипотезы.

Продуктовая гипотеза звучит так: меняем цветовую схему в приложении. Это должно увеличить количество пользователей, которые воспринимают приложение как приложение для женщин. Проверять будем на пользователях из России. Нужная метрика должна увеличиться на 34% в течение трех месяцев.

Кто в команде отвечает за сбор и обработку данных о поведении пользователей?

Аналитик

то в команде отвечает за разработку клиентской части интерфейса?

backend-разработчик

Что такое баг?

Это программа которая работает с ошибками

Алгоритмы

Что такое алгоритм?

Алгоритм — это **точно определённая инструкция, последовательно применяя которую к исходным данным, можно получить решение задачи**. Для каждого алгоритма есть некоторое множество объектов, допустимых в качестве исходных данных. Например, в алгоритме деления вещественных чисел делимое может быть любым, а делитель не может быть равен нулю.
Алгоритм служит, как правило, для решения не одной конкретной задачи, а некоторого класса задач.

Что такое псевдокод?

Псевдокод — это построчное неформальное описание кода будущей программы. Он полезен, когда нужно:

- **Описать работу алгоритма**. С помощью псевдокода можно объяснить, где и как в программе появляется определенная структура, механизм или прием.
- **Объяснить устройство программы пользователям, которые плохо разбираются в разработке**. При написании программы используется строгий синтаксис — иначе она не смогла бы правильно работать. Людям, особенно новичкам, проще воспринимать код на более простом и естественном языке, где понятна задача каждой строки.
- **Работать над кодом в команде**. Сеньор-разработчики часто используют псевдокод для решения сложных проблем,

с которыми сталкиваются мидлы и джуны, или просто чтобы объяснить свои действия.

Псевдокод будет действительно полезен и упростит разработку, если его правильно написать. Разберем основные правила работы с ним.

Что такое блок-схема и какие типы элементов у нее бывают?

Блок-схема — распространённый тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.

Приведите пример операции ввода.

Ввод какой то информации пользователем

Приведите пример операции вывода.

Вывод информации который ожидает пользователь от введенных донных

Приведите пример условия.

К примеру есть список и идем в магазин и проверяем если в магазине есть то что есть в списке мы покупаем иначе мы идем домой или в другой магазин

Приведите пример алгоритма с циклом.

К примеру нам нужно пробежать 2 км вокруг стадиона 1 круг стадиона равен 200 метров пробежали 1 круг проверяем равен ли 1 круг 2 км если нет бежим ещё пока не будет равен 2 км

Урок 2

Python — это компилируемый или интерпретируемый язык?

Python является **интерпретируемым** языком, то есть он выполняет код построчно. Если в коде программы присутствуют ошибки, она перестает работать. Это позволяет программистам быстро найти ошибки в коде. Простой в использовании язык. Python использует слова, подобные словам английского языка. В отличие от других языков программирования, в Python не используются фигурные скобки. Вместо них применяется отступ. Язык с динамической типизацией.

Какая разница между компилятором и интерпретатором?

Основная разница между ними состоит в следующем:
Компилятор обрабатывает сразу весь программный код после чего выдаёт результат, интерпретатор же берет одну инструкцию, преобразует и выполняет ее, а затем берет следующую инструкцию.
Компилятор генерирует отчет об ошибках после преобразования, а интерпретатор при нахождении ошибки прекращает свою работу.
Компилятор по сравнению с интерпретатором требует больше времени для анализа и обработки программного кода.

Какая типизация относится к Python: динамическая, статическая, сильная (строгая), слабая?

Python - язык с **динамической типизацией**. Это означает, что с определенным **типом** связывается не переменная, а ее значение. Если бы **Python** был языком со **статической типизацией**, мы бы не смогли сделать так: a= 1 a=' a' a= SomeClass (). **Динамическая типизация** - одна из причин популярности **Python**. Для начала, это просто удобно. Программа может менять **типы** переменных на лету, пользуясь их особенностями. a= (1, 1, 1, 3, 1, 1) a= list (set (a)) # [1, 3].

Что такое __pycache__?

__pycache__ - это каталог, содержащий файлы кэша байт-кода, которые автоматически генерируются python, а именно скомпилированные файлы Python

Переменные и типы

Что такое переменная?

В компьютерном программировании переменная или скалярная это **абстрактное место хранения в паре с соответствующим символическим именем, которое содержит некоторое известное или неизвестное количество информации, называемой значением**; или, проще говоря, переменная является контейнером для определенного типа данных (например, целое число, с плавающей точкой, строка и т. Д.). Переменная в конечном итоге может быть связана с адресом памяти или идентифицирована по нему.

Какие типы данных в Python вы знаете?

В языке Python выделяют несколько типов данных: **целые числа, числа с плавающей точкой (вещественные), строки, логический тип**. Тип каждой переменной может динамически изменяться по ходу выполнения программы. Определить, какой тип имеет переменная, можно с помощью команды type ().

Что такое преобразование типов? Приведите пример.

К примеру нам надо ввести через input() целое число нам нужно его преобразовать в целое число int(input())

Любые ли типы можно преобразовать? Что будет, если не получится преобразовать тип?

Не всё можно преобразовать к примеру “abcde” нельзя преобразовать в число

Расскажите про тип None.

NoneType — это встроенный тип данных в Python, который представляет отсутствие значения. Он предполагает, что переменная или функция не возвращает значение или что значение равно None или null. Ключевое слово None — это объект, тип данных класса NoneType. Мы можем присвоить None любой переменной, но мы не можем создавать другие объекты NoneType.

Что такое изменяемые и неизменяемые типы?

Изменяемые типы данных — это такие **типы**, элементы которых могут быть **изменены** или им могут быть присвоены новые значения. Списки, множества и словари являются примерами **изменяемых типов** данных в Python. **Неизменяемые типы** данных — это такие **типы**, значения которых не могут быть **изменены** после их создания.

Изменяемые объекты:

list, dict, set, byte array

Неизменяемые объекты:

*int, float, complex, string, tuple, frozenset (**неизменяемая версия** set), bytes*

Что такое хеширование и для чего оно применяется?

Хеширование (иногда «хэширование», англ. hashing) — **преобразование по детерминированному алгоритму входного массива данных произвольной длины в выходную битовую строку фиксированной длины**. Такие преобразования также называются хеш-функциями или функциями свёртки, а их результаты называют хешем, хеш-кодом или сверткой сообщения (англ. message digest).

Как хранятся переменные в Python?

Переменные в Python состоят из имени и значения. При этом они хранят в себе не само значение, а ссылку на его адрес в памяти.

Что возвращает функция id()?

Функция id() возвращет уникальный **идентификатор** переданного ей в качестве аргумента объекта. Этот **идентификатор** является адресом в памяти, по которому расположен сам объект. При каждом запуске программы этот **идентификатор** создается заново и будет для одного и того же объекта разным, за исключением случаев, когда у объектов есть свой постоянный уникальный **id**, как, например, у целых чисел от -5 до 256 - для них **id** будет одним и тем же при каждом вызове **функции id**().

Какие объекты являются хешируемыми в Python?

В Python объект является хешируемым, если у него есть __hash__ метод, который возвращает целое число, и __eq__ метод, который сравнивает два **объекта** на равенство. Неизменяемые типы данных **в Python**, такие как строки и числа, **являются хешируемыми** и могут быть элементами множества.

Булевые значения

Как оформляются комментарии в программе?

Комментарии оформляются при помощи # это комментарий

Какие значения может принимать переменная типа bool?

Переменная типа bool может принимать True либо False

Для чего бы вы использовали булев тип?

Для проверки условий или цикла

Как преобразуются в bool числа, строки и списки, по каким правилам, когда True/False ?

Переменная преобразуется в bool значение следующим образом

n = “hello”

string_bool = bool(n)

Что в Python является истиной, а что — ложью? Какие объекты считаются False?

Истиной является значение True а ложью False. False считается 0, -0, “”, ‘’, None, null, NaN, undefined

Примеры

- Что вернет

  `True + True + False`

  2

- Что вернет

  `bool("True")`

  True

- Что вернет

  `bool([])`

  False

- Что вернет

  `bool(-1)`

  True

- Что вернет

  `bool(None)`

  False

Числовые типы

Как округлить значение с точкой до целого? Пример: 3.34.

round(3.34)

Какой тип получится, если поделить целое на целое? Пример: 10/2.

10 / 2 = 5.0 float

Какой тип получится, если сложить целое с числом с плавающей точкой?

Будет float

Как проверить делимость на 2 нацело?

If num % 2 == 0 :

    Значит это число делится на 2

Что такое инкремент? Какой синтаксис используется для инкрементирования переменной?

Операторы инкремента (увеличения) и декремента (уменьшения) широко используются во многих языках программирования, таких как C++, чтобы увеличить или уменьшить значение переменной на единицу. Синтаксис этих операторов включает в себя двойные плюсы (++) или двойные минусы (—). Однако в Python этих операторов нет.

Вместо этого Python использует операторы += и -= для инкремента и декремента соответственно.

```
count = 0
count += 1 # Эквивалентно count = count + 1
print(count)  # Выведет: 1

count -= 1 # Эквивалентно count = count - 1
print(count)  # Выведет: 0
```

Как получить среднее арифметическое без цикла из списка чисел?

```
inp_lst = [12, 45, 78, 36, 45, 237.11, -1, 88]
sum_list = sum(inp_lst)
total = sum_list / len(inp_lst)
print(round(total,2))
```

Что такое декремент? Какой синтаксис используется для декрементирования переменной?

Операторы инкремента (увеличения) и декремента (уменьшения) широко используются во многих языках программирования, таких как C++, чтобы увеличить или уменьшить значение переменной на единицу. Синтаксис этих операторов включает в себя двойные плюсы (++) или двойные минусы (—). Однако в Python этих операторов нет.

Вместо этого Python использует операторы += и -= для инкремента и декремента соответственно.

```
count = 0
count += 1 # Эквивалентно count = count + 1
print(count)  # Выведет: 1

count -= 1 # Эквивалентно count = count - 1
print(count)  # Выведет: 0
```

Какие еще совмещенные операции вы знаете?

Примеры

- Что вернет выражение

  `round(2, 0)`

  2

- Что вернет выражение

  `round(2.0)`

  2

- Что вернет выражение

  `5 // 2`

  2

- Что вернет

  `2**3`

  8

- Что вернет

  `5 % 12`

  5

Логические операторы

Что такое логические операторы?

Это операторы сравнения

Чем отличается оператор `and` от оператора `or` ?

Оператор and это – “и” при условии оператор and проверяет чтобы первое значение было True и второе значение было True
А оператор or – “или” при условии оператор or проверяет чтобы первое значение было True а второе значение было False

Для чего нужен оператор `not` ?

переводит истину в ложь и наоборот.Обычно он используется с булевыми (логическими)значениями.При использовании с небулевыми значениями он возвращает false, если его единственный операнд может быть преобразован в true; в противном случае он возвращает true.

Что такое таблица истинности?

таблица, которая показывает, какие значения примет составное выражение при всех возможных наборах значений простых выражений, входящих в него. Сложное логическое выражение - логическое выражение, состоящее из одного или нескольких простых логических выражений (или сложных логических выражений), соединенных с помощью логических операций.

Примеры

- Чему равно выражение

  ```
  True and True and False
  ```

  False

- Чему равно выражение

  ```
  True or True or False
  ```

  True

- Чему равно выражение

  ```
  not not True
  ```

  True

- Чему равно выражение

  ```
  not True and not False
  ```

  False

Что такое ленивость в логических выражениях? (Вычисление по коротко схеме.

использует ленивый подход слева направо к оценке того, каким должен быть конечный результат. Вы всегда должны заботиться о том, чтобы правильно группировать свои оценки, иначе вы можете получить результаты, которых не ожидали.

С операторами `&&` оценка требует, чтобы все значения были истинными. Как только один элемент становится ложным, оценка останавливается и ответом будет `false`. Противоположное верно для `||`, где только один элемент должен быть истинным.

Операторы сравнения
Какие типы операторов сравнения есть в Python?

==
равно
!=
отличается
>
больше
<
меньше
>=
больше или равно
<=
меньше или равно

Какой синтаксис используется для двойных сравнений?

>=
больше или равно
<=
меньше или равно

Чем отличается оператор != и ==?

==
Равно
!=
Не равно

Какой тип данных получится при использовании оператора сравнения типа `4 > 2` ?

Тип bool True/False

Какой оператор позволяет проверить вхождение подстроки в строку?

Оператор in используется, чтобы **проверить**, содержится ли **подстрока в строке**: text = "Hello, world!" **substring** = "world". if **substring** in text: print("**Substring** found!") else: print("**Substring** not found!")

Примеры

- Что вернет выражение

  `4 >= 4`

  True

- Что вернет выражение

  `"py" in "Python"`

  False

- Что вернет выражение

  `"2" > "100"`

  True

- Что вернет выражение

  `True > 0`

  True

- Что вернет выражение

  `2 != 4 and 5`

  True

Чем отличается == от is?

Чем is **отличается** от == в Python. Оператор «==» возвращает True, если сравниваемые объекты имеют одинаковые значения. Оператор is возвращает True, если сравниваемые объекты, помимо того, что имеют одинаковые значения, еще и ссылаются на одну и ту же область памяти. Таким образом, is **отличается** от == в Python дополнительной проверкой на идентичность адресов в памяти, на которые ссылаются переменные.

Условные операторы

Чем отличается `elif` и `else`?
elif используется если условие if не выполняется elif можно использовать ни один раз else используется если ни одно из условий не выполнено

Чем отличается `elif` от `if`?
Проверка if всегда идет первой.
После оператора if должно быть какое-то условие: если это условие выполняется (возвращает True), то действия в блоке if выполняются.
С помощью elif можно сделать несколько разветвлений, то есть, проверять входящие данные на разные условия.

В каком порядке выполняются условные операторы?
Первый идет if если оно не выполняется идет elif если и оно не выполняется тогда выполняется else

В каком случае выполнится `elif`?
Если первое условие if не выполнено

Примеры

- Выполнится ли блок такого условия

  ```
  if True
  ```

  True

- Выполнится ли блок такого условия

  ```
  if 4 > 4
  ```

  False

- Выполнится ли блок такого условия

  ```
  if 1 is True
  ```

  ошибка

- Выполнится ли блок такого условия

  ```
  if '100' == 100
  ```

False

Стандарты кодирования

Что такое PEP-8? Приведите несколько примеров из правил.

С предсказуемым кодом приятно работать: сразу понятно, где импортированные модули, константа здесь или переменная и в каком блоке текущая строка. Для предсказуемости важны стиль и оформление, особенно в Python, который многое прощает разработчику.

Придерживайтесь этих стилей именования:

Тип	Рекомендация	Примеры
Функция	Одно или несколько слов в нижнем регистре, нижние подчеркивания для улучшения читаемости (snake case)	function, add_one
Переменная	Одна буква, слово или несколько слов в нижнем регистре, нижние подчеркивания для улучшения читаемости	x, connection, first_name
Класс	Одно или несколько слов с большой буквы без пробелов (camel case)	Image, UserData
Метод	Одно или несколько слов в нижнем регистре, нижние подчеркивания для улучшения читаемости	draw(), get_user_data()
Константа	Одна буква, слово или несколько слов в верхнем регистре, нижние подчеркивания для улучшения читаемости	PI, MAX_CONNECTIONS

Модуль	Короткое слово или слова в нижнем регистре, нижние подчеркивания для улучшения читаемости	module.py, user_data.py
Пакет	Короткое слово или слова в нижнем регистре без подчеркиваний	package, userdata

Проверка истинности без знаков равенства

Условия с булевыми значениями проверяются без оператора эквивалентности (==):

```
# Правильно

if this_is_true:
do_something()

if not this_is_false:
do_something_else()

# Неправильно

if this_is_true == True:
do_something()

if this_is_false == False:
do_something_else()
```

Что означает ошибка IndentError?

Эта ошибка возникает, когда оператор имеет ненужный отступ или его отступ не совпадает с отступом предыдущих операторов в том же блоке. Например, первый оператор в программе ниже имеет ненужный отступ:

Что означает ошибка NameError?

NameError возникает в тех случаях, когда вы пытаетесь использовать несуществующие имя переменной или функции.
В Python код запускается сверху вниз. Это значит, что переменную нельзя объявить уже после того, как она была использована. Python просто не будет знать о ее существовании.
Самая распространенная NameError выглядит вот так:
NameError: name 'some_name' is not defined.

Что означает принцип DRY (Don't Repeat Yourself)?

DRY – это аббревиатура английской фразы don’t repeat yourself, которая переводится как “не повторяйся”. Аббревиатура DRY (или “не повторяйся”) в мире программистов означает целый принцип (подход) к написанию программного кода, который считается базовым для всех начинающих программистов.

Что означает SRP (Single Responsibility Principle)?

SRP (Single Responsibility Principle) — это принцип, который гласит, что каждый класс или модуль должен иметь только одну ответственность и должен изменяться только по одной причине.

Это означает, что классы и модули должны быть разделены на отдельные компоненты, каждый из которых отвечает за выполнение конкретной задачи или функции. Это помогает улучшить читаемость, сопровождаемость и расширяемость кода, так как каждый компонент может быть изменен независимо от других.

Смысл SRP заключается в том, что код должен быть организован таким образом, чтобы каждая его часть отвечала только за свои задачи и не пересекалась с другими частями. Это повышает гибкость системы и облегчает ее модификацию в будущем.

Корректно ли называть переменную следующим образом и почему?

- `oblast = 10 #Некоректно`
- `a = "Test" #Некоректно`
- `name = 'Oleg' #Некоректно`
- `dict = {1 : 'one', 2 :  'two'} Некоректно`
- `CountData = 53 #Некоректно`
- `my_list = [1, 2, 3, 4, 5] #Коректно`
- `try = 'Try me' #Некоректно`

4. Урок 3. Списки и циклы

Списки

Что такое список?

Список (list) — это упорядоченный набор элементов, каждый из которых имеет свой номер, или **индекс**, позволяющий быстро получить к нему доступ.

Какие типы данных могут лежать в списках? А в одном списке?

Список может хранить любой тип данных

Сколько элементов в списке, если его последний элемент имеет индекс 2?

В списке имеются 3 элемента так как индекс начинается с 0.

[0], [1], [2],

Как добавить элемент в конец списка?

Список

my_list = [1, 2, 3, 4,]

my_list.append(5)

print(my_list)

>>>[1, 2, 3, 4, 5]

Как создать список?

My_list = [1, 2, 3, 4]

Как добавить элемент в середину списка?

```
x = [1, 2, 3, 12, 15, 18]
n = len(x)
i = n//2
x.insert(i, 50)
print(x)
```

Как объединить два списка?

list_a = [1, 2, 3, 4]

list_b = [5, 6, 7, 8]

list_c = list_a + list_b

print(list_c)

Что происходит с индексами списка при удалении элемента?
При удалении элементов из списка важно учитывать, что после удаления **индексы элементов, стоящих после удалённого, смещаются**. Например, был список: [a, b, c, d, e, f]. У e индекс 4 (счёт начинается с нуля). А теперь удалим b: [a, c, d, e, f]. У e индекс стал 3. Поэтому нельзя просто так брать список индексов и удалять по ним: индексы в этом списке станут указывать не туда, куда задумывалось.

Как проверить наличие элемента в списке?

Использовать функцию len() считает длину списка

Как отсортировать список алфавитно по убыванию?

my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

my_list.sort(reverse=True)

print(my_list)

Расскажи про разницу между insert, append, extend.

append

Это добавляет элемент в конец списка. Аргумент, переданный в функции append, добавляется в виде отдельного элемента в конце списка, а длина списка увеличивается на 1.

Синтаксис:
```
list_name.append(element)
```

index

Этот метод можно использовать для вставки значения в любую желаемую позицию. Он принимает два аргумента -элемент и индекс, по которому элемент должен быть вставлен.

Синтаксис:
```
list_name(index,element)
```

extend

Этот метод добавляет каждый элемент iterable (кортеж, строку или список) в конец списка и увеличивает длину списка на количество элементов iterable, переданных в качестве аргумента.

Синтаксис:

`list_name.extend(iterable)`

Как удалить элемент по индексу и значению?

Метод my_list.remove(“значение”)

Метод my_list.pop[2] удаление по индексу и возвращает значение

Оператор del my_list[2] удаляет по индексу но можно и по срезам

Как создать список из строки?

my_str = “Hello word!”

my_list = my_str.split()

print(my_list)

Как найти позицию элемента в списке?

```
fruits = ["apple", "orange","grapes","guava"]
print(fruits.index('orange')) # 1
```

- Что вернет выражение

  `sorted(["B", "A", "D"]) # A, B, D`

- Что вернет выражение

  `["A", "B", "C"][0] # A`

-

- Что вернет выражение

  `["AA", "BB", "CC"][1][1]`

  B

- Что вернет выражение

  `["A", "B", "C"].pop()`

  C

- Что вернет выражение

  `["A", "B", "C"][-1]`

  C

- Что вернет выражение

  `list("abc")`
  ['a', 'b', 'c']

Что такое copy? Зачем он нужен?

Метод copy принимает исходный объект коллекции в качестве входных данных и создает новый объект коллекции со ссылкой на элементы исходного объекта коллекции. Затем он возвращает ссылку на новый объект коллекции.

```
import copy


data = {1: 5, 2: 6, 3: 7, 4: 8}

print("Оригинальный словарь:")

print(data)

new_data = copy.copy(data)

print("Новый словарь:")

print(new_data)
```

Оригинальный словарь:

{1: 5, 2: 6, 3: 7, 4: 8}

Новый словарь:

{1: 5, 2: 6, 3: 7, 4: 8}

Что такое deepcopy?

Метод deepcopy рекурсивно создает копию каждого элемента объекта и не копирует ссылки.

```
import copy


data = {1: 5, 2: 6, 3: 7, 4: {8: 9}}

print("Оригинальный словарь:")

print(data)

new_data = copy.deepcopy(data)

print("Копия словаря:")

print(new_data)

data[4][8] = 99

print("Оригинал после изменения:")

print(data)

print("Копия после изменения:")

print(new_data)
```

```
Оригинальный словарь:

{1: 5, 2: 6, 3: 7, 4: {8: 9}}

Копия словаря:

{1: 5, 2: 6, 3: 7, 4: {8: 9}}

Оригинал после изменения:

{1: 5, 2: 6, 3: 7, 4: {8: 99}}

Копия после изменения:

{1: 5, 2: 6, 3: 7, 4: {8: 9}}
```

Списки — это итерируемые объекты. Что это значит и какие еще итерируемые объекты есть в Python?

Итерируемый объект (iterable) - это объект, который способен возвращать элементы по одному. Кроме того, это объект, из которого можно получить итератор. Примеры итерируемых объектов: все последовательности: **список, строка, кортеж словари файлы**. В Python за получение итератора отвечает функция iter (): In [1]: lista= [1, 2, 3] In [2]: iter (lista).

Срезы и индексы

К чему можно применять срезы?

Срезы (slices) — это удобный инструмент в Python для работы с итерируемыми объектами, такими как строки, списки и кортежи. Они позволяют легко извлекать часть объекта, не изменяя его исходное состояние.

Приведите пример индексов для получения первого и

```
a = [1, 2, 3, 4, 5]
ans = a[::len(a) -1]
print(ans)
```

Как получить последние три элемента?

```
a = [1, 2, 3, 4, 5]
print(a[-3:])
```

Как получить все нечётные элементы?

```
a = [1, 2, 3, 4, 5]
print(a[::2])
```

- Что вернет

  `"abcde"[:]`

  `abcde`

- Что вернет

  `"abcde"[:2]`

  ab

- Что вернет

  `"abcde"[2:2]`

[] или пустой список

- Что вернет

  `"abcde"[2:]`

  cde

- Что вернет

  `"abcde"[-2:]`

  de

Циклы

for и while

Для чего используется range?

Функция range () в Python очень часто используется **для создания коллекции из последовательных чисел на лету**, например 0, 1, 2, 3, 4. Это очень практично, поскольку готовую последовательность чисел можно использовать для индексации коллекций или, например, для итерации в циклах.

Сколько итераций у цикла for i in range(0,3)?

Будет 3 итерации 0, 1, 2,

Как вывести числа от 1 до 100 в цикле?

for i in range(1, 101)

Что будет, если запустить перебор по пустому списку?

Не чего не произойдет как список у нас пестой

Для чего используется enumerate?

enumerate() используется для нумерации списка

letters = [“Alpha”, “Bravo”, “Charlie”, “Delta”, “Echo”]

for i, letter in enumerate(letters, start=1):

    print(i, letter)

В каких случаях целесообразнее использовать цикл while?

цикл N раз используется когда когда нужно сделать что-то N раз. цикл условие когда не известно длинна чего нибудь к примеру: нц пока слева забор. вперёд.

Для чего используется оператор continue?

Оператор **continue** нужен для пропуска 1 итерации в цикле, чаще всего за каким либо условием.

Для чего используется оператор break?

Для приостановки цикла

Когда его целесообразно использовать?

При использовании бесконечного цикла while True

Какое количество итераций будет выполнено в таком цикле?

- `for i in range(10)` 10 от 0 до 9
- `for i in range(-5, -1)` 4 от -5 до -2
- `for i in range(-1, -5)` не чего не выведет
- `list = [1, 2, 3]`
- `for i in list  1, 2, 3`
- `for i in range(len(list))` 0, 1, 2

- `for i in range(len(list), 1) не чего не выведет`

Для чего нужен оператор else в циклах?

Завершение цикла естественным путем

Урок 4. Строки и словари

Как получить символ строки по индексу?

string = “Hello word!”

print(string[2] # Выведет l

Как получить длину строки?

string = “Hello word!”

print(len(string))

Как удалить пробелы и другие непечатаемые символы в начале и/или конце строки?

Функция strip() удаляет пробелы или символы вокруг строки

Как посчитать, сколько раз входит подстрока в строку?

Используется метод count()

string_ = “Hello word!”

str_ = string_.count(“l”)

print (str_) вернет 2

Как найти позицию подстроки в строке?

string_ = “Hello word!”

str_ = string_.find(“l”)

print (str_) вернет 2

Как разделить строку на элементы по заданному символу?

string_ = “Hello word!”

str_ = string_.replace(“!”, “?”)

print(str_)

Как удалить подстроку из строки?

string_ = “Hello word!”

str_ = string_.replace(“l”, “”)

print(str_)

Как превратить строку в список, а список — в строку?

string_ = “Hello word!”

str_1 = string.split()

print(str_)

str_2 = “ ”.join(str_1)

print(str_2)

Как изменится строка, состоящая из 1, при вызове метода replace?

При вызове метода `replace` для строки, состоящей из одного символа, результат будет зависеть от того, какой символ был заменен и какой символ подставляется.

Если символ, который нужно заменить, равен нулю (0), то строка останется неизменной. Если символ, который нужно заменить, не равен нулю, то результат будет строковым представлением числа, соответствующего символу, который был заменен.

Например, вызов метода `replace('1', '2')` для строки `"1"` вернет строку `"2"`, а вызов метода `replace(0, '3')` вернет пустую строку.

Как перевести число в строку?

```
num = 10

str_1 = str(num)

print(str_1)
```

Как проверить, является ли строка последовательностью цифр?

```
str_1 = "01234"

str_2 = str_1.isdigit()

print(str_2)
```

Как проверить, является ли строка последовательностью букв?

```
str_1 = "ABCD"

str_2 = str_1.isalpha()

print(str_2)
```

Как проверить, является ли строка последовательностью букв и цифр?

Для проверки строки на то, является ли она последовательностью букв и чисел, можно использовать регулярное выражение. В Python для этого можно использовать модуль re. Вот пример кода:

```
import re
```

```
string = "1234567890"

if re.match(r"[a-zA-Z\d]+", string):

    print("Строка содержит буквы и числа")

else:

    print("Строка не содержит букв и чисел")
```

Как перевести строку в нижний/верхний регистр?

```
string_ = “Hello word!”

str_1 = string_.lower()

print(str_1)

str_2 = string_.upper()

print(str_2)
```

Что такое экранирование и зачем оно нужно?

Экранирование - это процесс замены специальных символов в строке на их соответствующие escape-коды. Это может быть полезно, когда вы работаете с текстом, содержащим специальные символы, такие как кавычки, пробелы или символы табуляции.

Например, если вы хотите вставить кавычку в строку, вы можете использовать одинарную кавычку ’ вместо обычных кавычек ". Однако, если вы используете одинарную кавычку в строке, Python интерпретирует ее как конец строки, а не как часть строки. Чтобы избежать этого, вы можете экранировать одинарную кавычку с помощью обратного слеша . Таким образом, строка с одинарными кавычками будет выглядеть так: ".

Кроме того, экранирование может использоваться для работы с символами табуляции и пробелами. Например, если вы хотите добавить символ табуляции в строку, вам нужно экранировать его с помощью символа обратной косой черты \t.

В целом, экранирование используется для корректной обработки специальных символов в строках, что позволяет избежать ошибок в коде и обеспечить правильную работу программы.

Примеры

- Что вернет

  `"01234".isdigit()`

  True

- Что вернет

  `"ABCD".isalpha()`

  True

- Что вернет

  `"abc".isupper()`

  False

- Что вернет

  `str(100 + 00)`

  100

- Что вернет

  `"a" in "abc"`

  True

- Что вернет

  `"B" in "abc"`

  False

- Что вернет

  `"a".count("abc")`

  1

- Что вернет

  `len("b o o".split())`

Словари

Для чего используются словари?

Словари в программировании используются для хранения пар ключ-значение. Ключи могут быть любого типа данных, а значения могут быть любыми другими типами данных. Словари позволяют быстро искать значения по ключам и добавлять новые значения. Они также могут использоваться для организации данных и упрощения работы с ними.

Как получить коллекцию ключей из словаря?

Для получения коллекции ключей словаря в Python можно использовать метод keys() или dict.keys().

Пример:

my_dict = {‘key1’: ‘value1’, ‘key2’: ‘value2’}

keys = my_dict.keys()

print(keys) # [‘key1’, ‘key2’]

Или:

keys2 = dict.keys(my_dict)

print(keys2) # [‘key1’, ‘key2’]

Как получить коллекцию значений из словаря?

Для получения коллекции значений из словаря в Python, можно использовать методы values() или dict().values().

Вот пример:

```
my_dict = { 'key1': 1, 'key2': 2, 'key3': 3 }
values = list(my_dict.values())
print(values)  # [1, 2, 3]
```

Как удалить элемент из словаря?

Оператор del используется для удаления элемента по ключу из словаря. Например:

```
my_dictionary = {'apple': 10, 'banana': 20}

del my_dictionary['apple']

print(my_dictionary) # {'banana': 20}
```

В этом примере мы удалили ключ ‘apple’ из словаря my_dictionary и сохранили результат в переменной my_dictionary.

Какие типы данных могут быть ключами в словаре?

Ключом может быть произвольный неизменяемый тип данных: целые и действительные числа, строки,
кортежи. Ключом в словаре не может быть множество,
но может быть элемент типа frozenset: специальный тип данных, являющийся аналогом типа set, который нельзя изменять после создания.

Какие типы данных могут быть значениями в словаре?

Любые типы данных

Что будет выведено при переборе словаря циклом?

При переборе словаря с помощью цикла for в Python, на каждой итерации цикла будет выводиться значение ключа и значение соответствующего ему значения. Например:

```
d = {'a': 1, 'b': 2, 'c': 3}

for key, value in d.items():

    print(key, value)
```

Результат выполнения данного кода будет следующим:

```
a 1
b 2
c 3
```

Что означает ошибка KeyError?

Ошибка возникает когда хотим вывести несуществующий ключ словаря

Какой оператор позволяет проверить вхождение ключа в словарь?

Для проверки вхождения ключа в словарь можно использовать оператор `in`. Например:

```
d = { 'apple': 1, 'orange': 2 }
if 'apple' in d:
    print("Apple is in the dictionary.")
```

- Что вернет

  `1 in {1:"one", 2: "two"}`

  True

- Что вернет

  `{1:"one", 2: "two"}["one"]`

  Ошибку

- Что вернет

  `{1:"one", 2: "two"}.pop()`

  Ошибку

- Что вернет

  `{1:"one", 2: "two"}.pop(2)`

  two

- Что вернет

  `2 in {1:"one", 2: "two"}`

  True

- Что вернет

```
len ({1:"one", 2: "two"}.get(1) )
```

3

Как организовано хранение данных в словаре?

Хранение данных в словарях организовано следующим образом:

– Каждый элемент словаря представляет собой пару ключ-значение, где ключом может быть любой допустимый тип данных, а значением может быть любой другой допустимый тип данных.
– Ключи словаря автоматически упорядочиваются по алфавиту, поэтому если вы создадите словарь с ключами ‘a’, ‘b’, ‘c’, то порядок хранения ключей будет ‘a’, ‘b’ и ‘c’.
– Для добавления нового элемента в словарь используется операция ‘+=’, а для удаления элемента – оператор ‘del’.
– Значения словаря могут быть изменены с помощью операции ‘update()’, которая заменяет все существующие значения новыми значениями.
– Доступ к элементам словаря осуществляется с помощью квадратных скобок, например: my_dict[‘ключ’] = значение.

Что такое хеш-таблица?

Хеш-таблица (Hash table) - это структура данных, используемая для хранения пар ключ-значение в виде массива, где ключ используется для доступа к значению. Она позволяет быстро искать и изменять данные, используя хеш-функцию, которая преобразует ключ в целочисленное значение (хеш).

Хеш-таблицы часто используются для хранения информации в базах данных, таких как пароли, электронные письма и другие конфиденциальные данные. Они также используются в программировании для быстрого доступа к данным в структурах данных, например, словарях (dictionaries).

Что такое метод get и как он работает?

Метод `get` - это метод, который используется для извлечения значения по заданному ключу из словаря Python. Он возвращает значение, если ключ найден, иначе возвращает None, если ключ не найден.

Пример использования метода `get`:

```
d = {'name': 'John', 'age': 30}

value = d.get('name')

print(value) # 'John'

value = d.get('age')

print(value) # 30
```

В этом примере метод `get` используется для извлечения значений по ключам `'name'` и `'age'`. Если ключ `'name'` найден в словаре `d`, то метод `get` вернет его значение `'John'`. Если же ключ `'age'` не найден в словаре, то метод `get` вернет значение `None`.

Что такое метод setdefault и как он работает?

Метод `setdefault()` - это метод словаря в Python, который позволяет установить значение для отсутствующего ключа и вернуть значение этого ключа. Если ключ уже существует в словаре, то этот метод ничего не делает и просто возвращает текущее значение ключа. Если же ключа нет в словаре, то он добавляется в словарь и ему присваивается указанное значение.

Метод `setdefault()` является более эффективным, чем использование оператора `dict.get()` вместе с оператором `dict.setdefault()`, так как он не вызывает метод `dict.get()`, если ключ уже существует.

Пример использования `setdefault()`:

```
# Создаем словарь
my_dict = {"name": "John", "age": 30, "city": "New
York"}

# Проверяем наличие ключа 'city' в словаре
if "city" in my_dict:
    # Если ключ 'city' существует, то возвращаем
текущее значение
    city = my_dict["city"]
else:
    # Иначе добавляем ключ 'city' со значением 'New
York' в словарь
    my_dict["city"] = "New York"
```

```
# Выводим значение ключа 'city'
print(my_dict)
print(city)
```

Какие типы данных нельзя использовать в качестве ключей словаря?

Изменяемый тип данных в Python всего 3: список, словарь и множество. Их нельзя использовать в качестве ключей словоря

Урок 5. Функции и область видимости

Функции

Что такое функция?

В Python функция - это блок кода, который может быть вызван из других частей программы для выполнения определенной задачи. Функции могут принимать аргументы (параметры), изменять значения переменных и возвращать результат своей работы.

Пример функции в Python:

```
def my_function(arg1, arg2):

    # код функции

    return result
```

В этом примере функция my_function принимает два аргумента arg1 и arg2 и возвращает результат. Внутри функции можно использовать переменные, которые были определены в основной части программы, а также создавать новые переменные и изменять их значения.

Вызов функции в Python осуществляется с помощью ключевого слова “def” и имени функции. Например:

```
result = my_function(5, 7)
```

Здесь мы вызвали функцию my_function с аргументами 5 и 7 и сохранили результат ее работы в переменной result.

Зачем нужно писать функции?

Функции используются в программировании для структурирования кода и повышения его эффективности. Они позволяют разбить большой код на более мелкие и понятные блоки, что упрощает его чтение, понимание и поддержку. Кроме того, функции позволяют повторно использовать код, что сокращает время на написание новых программ и повышает их надежность.

Какое ключевое слово используется для создания функции?

Вызов функции в Python осуществляется с помощью ключевого слова “def” и имени функции. Например:

```
def my_function(arg1, arg2):
     # код функции
    return result


result = my_function(5, 7)
```

Какие названия функций нельзя использовать?

В Python есть несколько правил, касающихся названий функций. Вот некоторые из них:

1. Нельзя использовать зарезервированные слова Python, такие как `def`, `if`, `else`, `for` и т. д. в качестве имен функций.
2. Имена функций должны быть существительными в единственном числе и начинаться с буквы.
3. Имена функций не должны содержать пробелов или знаков препинания.
4. Имена функций рекомендуется выбирать так, чтобы они отражали их назначение или логику работы.

Что такое аргументы и параметры функции?

Аргументы функции - это данные, которые функция принимает при вызове. Параметры функции - это именованные аргументы, которые можно передавать в функцию при ее вызове. Они используются для передачи данных в функцию и изменения ее поведения.

Что такое позиционные аргументы?

Позиционные аргументы - это аргументы, которые имеют определенный порядок в вызове функции. В Python позиционные аргументы указываются после имени функции и разделяются запятыми. Например, if func(a, b, c): будет позиционным аргументом, так как значения a, b и c передаются в определенном порядке.

Что такое именованные аргументы?

Именованные аргументы - это аргументы функции, которые могут быть переданы в функцию с использованием именованных параметров. Они позволяют передавать данные в функцию в любом порядке и с любыми именами, которые вы хотите. Именованные параметры указываются после имени функции с двоеточием и именем аргумента. Например, def func(a: int, b: str, c: float) -> None: будет именованным аргументом, так как имена a, b и c могут быть любыми.

Что сначала пишется: именованные или позиционные аргументы?

Именованные аргументы обычно указываются в функции после ее имени, а позиционные аргументы - в круглых скобках. Поэтому, сначала пишутся позиционные аргументы, а затем именованные аргументы.

Как задавать значения по умолчанию?

Для задания значений по умолчанию в функциях в Python можно использовать оператор присваивания с ключевым словом default.

Например, если у нас есть функция, которая принимает два аргумента и возвращает их сумму, но если один из аргументов не был передан, то мы хотим использовать значение по умолчанию 0:

```
def add(x, y, default=0):

    return x + y + default
```

Как вернуть значение из функции в тело программы?

Для возврата значения из функции в программу можно использовать оператор return. Например:

```
def sum_of_multiples(n):
    for i in range(2, n):
        if i % 3 == 0 or i % 5 == 0:
            return i
    return 0

sum_of_multiples(1000)
```

Что вернет функция, у которой нет return в коде?

Если функция не содержит оператора return, то она ничего не возвращает.

Что такое докстринги и зачем писать документацию?

Докстринги (docstrings) - это комментарии, которые описываются функцию в языке программирования Python. Они помогают разработчикам понять, как работает функция и какие аргументы она принимает.

Написание документации необходимо для того, чтобы пользователи могли легко понять, как использовать функции и классы в вашем программном обеспечении. Документация может быть написана в различных форматах, таких как “”” “””, HTML, Markdown или XML, и

может содержать информацию о функциональности, синтаксисе, примерах использования и т.д.

Как принять заранее неизвестное количество аргументов?

Для приема заранее неизвестного количества аргументов в Python можно использовать кортежи. Например:

```
def my_function(*args):
    # args - кортеж, содержащий все аргументы
    print(args)
```

Здесь *args - это кортеж, который содержит все переданные аргументы. Вы можете передать любое количество аргументов, и они будут сохранены в кортеже args.

Что такое чистые функции?

Чистые функции - это функции, которые не изменяют состояние своего окружения. Они возвращают результат на основе входных данных и не имеют побочных эффектов. Чистые функции являются важным принципом функционального программирования и используются для создания более эффективных и безопасных программ.

Что произойдет, если переопределить функцию как число, а потом ее вызвать?

Если вы переопределите функцию как число и затем вызовете ее, то произойдет ошибка. Это связано с тем, что функции в Python должны быть определены с помощью ключевого слова def, а числа не могут быть использованы в качестве аргументов функции.

Что вернет return без операнда?

Если в функции отсутствует оператор return без операндов, то функция вернет None.

Чем отличается args от kwargs?

Аргументы (args) и ключевые аргументы (kwargs) являются двумя разными способами передачи параметров в функцию в Python.

args - это список или кортеж, в котором передаются значения параметров в функции. Каждый элемент списка или кортежа представляет собой значение параметра. Например, если у вас есть функция add(a, b), вы можете вызвать ее следующим образом: add(3, 4). В этом случае args будет содержать значения 3 и 4.

kwargs - это словарь или запись, в которой передаются параметры в функции. Ключи словаря - это имена параметров, а значения - их значения. Например, если у вас есть функция add(*args), вы можете передать ей несколько параметров, разделенных звездочкой ()*. В этом случае kwargs будет содержать ключ ‘a’ со значением 3 и ключ ‘b’ со значением 4.

Основное отличие между args и kwargs заключается в том, что args передает значения параметров по порядку, а kwargs передает их в произвольном порядке. Кроме того, kwargs позволяет передавать произвольное количество параметров, в то время как args может принимать только фиксированное количество параметров.

Можно ли переименовать функцию?

Да, можно переименовать функцию. Для этого нужно изменить имя функции в исходном коде программы, а также в любых других местах, где эта функция используется.

Можно ли положить функцию в список или словарь?

Да, функции в Python можно положить в список и словарь. Вот пример:

```
def add_sub(x, y):
    return x + y

mylist = [add_sub]  # создаем список из одной функции
print(mylist)  # выводим список
```

```
mydict = {
    'add': add_sub,
    'sub': lambda x, y: x - y
}  # создаем словарь с двумя функциями
print(mydict)  # выводим словарь
```

Сколько раз в функции может быть выполнен return?
В функции Python может быть только один оператор return. Если в функции несколько операторов return, то только последний оператор будет иметь эффект. Все предыдущие операторы return будут проигнорированы.

Область видимости

Что такое область видимости? Расскажите, что знаете.
Область видимости - это область, в которой можно использовать переменные и функции, которые были определены в этой области. В Python есть три уровня области видимости: глобальный, локальный и анонимный.

Глобальная область видимости - это все переменные и функции, определенные вне функций. Они доступны во всем коде программы.

Локальная область видимости - это переменные и функции, определенные внутри функций. Они доступны только внутри этой функции и не могут быть использованы за ее пределами.

Анонимная область видимости - это область видимости, которая создается при использовании анонимных функций. Эти функции не имеют имени и могут быть использованы только в том месте, где они были созданы.

Какие области видимости вы знаете?

Глобальная область видимости

Локальная область видимости

Анонимная область видимости

Зачем область видимости нужна?

Область видимости нужна для того, чтобы избежать ошибок, связанных с использованием переменных и функций, которые были определены вне этой области. Если переменная или функция определены вне функции, то они будут доступны во всей программе, что может привести к ошибкам при использовании их внутри функции.

Что такое оператор global и почему лучше его не использовать?

Оператор global в Python используется для изменения значения переменной, которая была объявлена вне функции. Однако, использование этого оператора может привести к проблемам, связанным с отладкой программы и безопасностью данных.

Во-первых, оператор global может привести к изменению значения переменной внутри функции, что может вызвать проблемы с согласованностью данных. Например, если переменная была изменена внутри функции, то ее значение может измениться и вне функции, что приведет к неожиданным результатам.

Кроме того, использование operator global может снизить безопасность данных в программе. Если функция изменяет значение переменной, то это может привести к тому, что другие функции также будут изменять ее значение, что может нарушить работу программы.

Поэтому, чтобы избежать проблем с согласованностью данных и безопасностью программы, лучше не использовать оператор global в Python. Вместо этого, следует использовать локальные переменные и функции для работы с данными внутри функции.