

1. Основы Python

Вопрос 1. К какому уровню относится язык программирования Python?

Ответ: к высокоуровневому языку программирования

Вопрос 2. Какую типизацию имеет язык Python?

Ответ: язык Python имеет неявную, динамическую и строгую типизацию

Вопрос 3. Объясните, что представляет собой неявная типизация?

Ответ: неявная типизация – это когда тип переменной заранее не объявляется. Но здесь надо помнить, что **в Python переменные не хранят объект, а лишь ссылаются на ячейку памяти, где он записан.**

LiveCoding: какие средства в Python существуют, чтобы обойти неявную типизацию, приведите примеры?

В синтаксисе Python предусмотрена аннотация типов, например, `a: int = 2`, `b: float = 2.5`. Также существует библиотека `typing` для аннотации нескольких типов данных.

Вопрос 4. Объясните, что представляет собой динамическая типизация?

Ответ: динамическая типизация – это когда тип присваивается непосредственно при выполнении программы. Это означает, что переменная может менять свой тип в зависимости от последнего присвоенного ей значения. **В Python переменные не хранят объект, а лишь ссылаются на ячейку памяти, где он записан.**

LiveCoding: Приведите пример динамической типизации?

```
a = 3
print(type(a))
a = "string"
print(type(a))
```

Вопрос 5. Что означает символ «`=`»?

Ответ: это оператор присваивания, он означает ссылку на ячейку в памяти компьютера. То есть переменная ссылается на какой-нибудь объект в памяти компьютера

LiveCoding: Как проверить на какую ячейку памяти ссылается переменная?

С помощью функции `id`

```
a = 8.2
print(id(a))
```

Вопрос 6. Что такое каскадное присваивание?

Ответ: это присваивание вида `a = b = c = 0`

Вопрос 7. Как узнать какой тип данных, на которые ссылается переменная?

Ответ: С помощью функции type.

LiveCoding: напишите несколько переменных, которые ссылаются на объекты разных типов и проверьте их типы?

```
a = 1
b = "Hello!"
print(type(a), type(b))
```

Вопрос 8. Что такое сильная типизация?

Ответ: это когда нельзя проводить операции над объектами разных типов

LiveCoding: вызовите ошибку из-за сильной типизации?

```
a = 'корова' + 1.5
print(a)
```

Вопрос 9. Объекты в ячейки памяти можно изменять?

Ответ: Это зависит от типа данных. В Python есть изменяемые типы данных, и неизменяемые

Вопрос 10. Перечислите изменяемые типы данных Python?

Ответ: **множества set{}, списки list[], словари dict{}**.

Также есть изменяемые типы данных в стандартных библиотеках Python. Например, в библиотеки collections есть тип данных deque, или же тип array в стандартной библиотеке array

LiveCoding: напишите примеры изменяемых типов данных и присвойте им переменные

Вопрос 11. Какая особенность есть у изменяемых типов данных Python с точки зрения хеширования?

Ответ: для таких данных не работает hash, т.е. кодирование которое преобразует любое значение в целое число.

Вопрос 12. Перечислите неизменяемые типы данных Python?

Ответ: целые числа int(), числа с плавающей точкой (запятой) float(), комплексные числа complex(), строковый тип данных str(), кортежи tuple(), булевый тип bool, нулевой тип NoneType

LiveCoding: напишите примеры неизменяемых типов данных и присвойте им переменные

Вопрос 13. Как называются объекты, которые способны возвращать элементы по одному?

Ответ: такие объекты называются итерируемые

LiveCoding: перечислите итерируемые объекты

списки list[], строковый тип данных str(), кортежи tuple(), множества set{}, словари dict{}, файлы, генераторы

Вопрос 14. В какие типы данных можно передавать любой итерируемый объект?

Ответ: Только итерируемые объекты можно передавать в списки `list[]` и кортежи `tuple()`.

Вопрос 15. Можно ли производить арифметические операции для разных типов данных в Python?

Ответ: арифметические операции применимы только для данных, которые имеют одинаковые типы

LiveCoding: напишите примеры арифметических операций объектов одинаковых типов данных.

Вопрос 16. Перечислите все арифметические операции в Python?

Ответ: сложение, вычитание, умножение, деление, целочисленное деление, возведение в степень, деление с остатком.

LiveCoding: приведите примеры всех арифметических операций в Python?

Вопрос 17. Какие операции бывают в Python помимо арифметических?

Ответ: логические операции, битовые операции

Вопрос 18. Какие приоритеты выполнения ВСЕХ АРИФМЕТИЧЕСКИХ операций?

Ответ: скобки имеют самый высокий приоритет, далее возведение в степень, далее арифметические операции вначале умножение, деление, целочисленное деление, взятие остатка от деления, потом сложение, вычитание, далее все битовые операции и в конце логические операции

LiveCoding: вычислите `27 ** 1/3`

Вначале 27 возводится в степень 1, а только потом делится на 3. Ответ -9

Вопрос 19. Какие приоритеты выполнения ЛОГИЧЕСКИХ операций?

Ответ:

1) самый высокий приоритет у сравнения, проверка идентичности, проверка вхождения ``==, !=, >, >=, <, <=, is, is not, in, not in``

2) Логическое НЕ ``not``

3) Логическое И ``and``

4) Логическое ИЛИ ``or``

Вопрос 20. Какая разница между `<=>` и `<==>`?

Ответ `<=>` - это оператор присваивания, который говорит, что переменная ссылается на определенный объект в ячейки памяти компьютера. `<==>` - символ (оператор) сравнения, он сравнивает левую и правую часть выражения, и если они совпадают возвращает `True`, если нет, то `False`

Вопрос 21. Как можно округлять числа в Python?

Ответ: метод `round(var, x)`, где `var` - переменная, которую мы хотим округлить, а `x` - это сколько знаков мы хотим видеть после запятой, метод `:.2f` работает только для f-строк, метод с использованием библиотеки `Decimal`

LiveCoding: приведите примеры округления чисел

Вопрос 22. Что такое целочисленный тип данных?

Ответ: `int` (сокращенно от `integer` - целое число) - это все отрицательные, положительные целые числа и ноль.

LiveCoding: приведите примеры как можно упростить читаемость большого числа типа `int`

```
432246123 == 432_246_123
```

Вопрос 23. Что такое бинарный вид целого числа?

Ответ: любое целое число можно представить в бинарном виде, т.е. в виде нулей и единиц.

LiveCoding: представите число 124 в бинарном виде?

```
a = 124
print(bin(a))
```

Вопрос 24. Для чего иногда нужно представлять целые числа в бинарном виде?

Ответ: К целым числам можно применять битовые операции - они позволяют менять число путем включения и отключения бит (т.е. нулей и единиц в бинарном виде). Это позволяет оптимизировать память, и программа будет работать быстрее. Т.е. это один из способов снижения времени выполнения алгоритма.

LiveCoding: покажите любую операцию для бинарного вида целого числа?

Допустим надо включить бит №3 у числа 100

```
print(100 | (1<<3))`
108
```

Вопрос 25. Какой тип данных соответствует ничему, пустому типу?

Ответ: `NoneType`

Вопрос 26. Какой тип данных соответствует действительному числу из математики?

Ответ: `float` - это тип с плавающей точкой

Вопрос 27. Что такое списки в Python?

Ответ: Объект `list = [...]` - это упорядоченная изменяемая коллекция, которая служит для группировки каких-нибудь данных. С точки зрения структур данных, представляет собой динамический массив

Вопрос 28. Что значит упорядоченная коллекция?

Ответ: Упорядоченная означает, что к каждому элементу списка можно обратиться по индексу или по ключу.

Вопрос 29. Что происходит в ячейки памяти, если изменить элементы в списке?

Ответ при изменении элементов в списке в ячейки памяти **не создается** новый объект, а остается прежний, только измененный

Вопрос 30. Могут ли быть разные типы данных внутри списка?

Ответ: да, могут

LiveCoding: напишите список, у которого элементы - разные типы данных

Вопрос 31. Какая есть функция в Python для ввода данных и какой тип данных она возвращает?

Ответ: функция `input()`. Вне зависимости от того, что вы ввели, эта функция возвращает строку (тип `str`)

Вопрос 32. Как можно задать список, используя функцию для ввода данных?

Ответ:

```
lst = input().split()
```

Вопрос 33. Произведите замену значения элемента с индексом 1 на 666 в списке `num = [1, 2, 4]` ?

Ответ:

```
num = [1, 2, 4]
num[1] = 666
print(num)

[1, 666, 4]
```

Вопрос 34. Дан список `winter = [-10, -5, -20, -21, -4, -2]`. Вывести его элемент, имеющий минимальное значение.

Ответ:

```
winter = [-10, -5, -20, -21, -4, -2]
print(min(winter))
-21
```

Вопрос 35. Дан список `summa = [7, 8, 10]`. Найти сумму всех его элементов.

Ответ:

```
summa = [7, 8, 10]
print(sum(summa))
25
```

Вопрос 36. Дан список `lst = [1, 3, 4, 6]`. Произвести срез элементов так, чтобы можно было вывести в консоль список `[3, 4]`.

Ответ: необходимо сделать срез списка.

```
lst = [1, 3, 4, 6]
print(lst[1:3])
[3, 4]
```

Вопрос 37. Дан список `marks = [2, 3, 4, 5, 3]`. С помощью среза заменить элемент со значением «4» на «хорошо», элемент со значением «5» на «отлично». Должно получиться `[2, 3, 'хорошо', 'отлично', 3]`.

Ответ:

```
marks = [2, 3, 4, 5, 3]
marks[2:4] = ["хорошо", "отлично"]
print(marks)
[2, 3, 'хорошо', 'отлично', 3]
```

Вопрос 38. Что такое конкатенация для списков?

Ответ: это объединение списков в третий список с помощью «+»

LiveCoding: приведите пример конкатенация для списков `list_1 = ['one', 'two', 'three']` и `list_2 = ['one', 2, 3.0]`?

Ответ:

```
list_1 = ['one', 'two', 'three']
list_2 = ['one', 2, 3.0]
list_3 = list_1 + list_2
print(list_3)
```

Результат: `['one', 'two', 'three', 'one', 2, 3.0]`

Вопрос 39. Как можно преобразовать строку в список и что в этом случае получится?

Ответ:

```
name = 'Василий'
print(list(name))
Результат: ['В', 'а', 'с', 'и', 'л', 'и', 'й']
```

Вопрос 40. Как добавить элемент в конец списка?

Ответ: с помощью `list.append(_x_)`

```
list_1 = ['dog', 'cat']
list_1.append('pig')
print(list_1)
```

Результат: ['dog', 'cat', 'pig']

Вопрос 41. Как можно расширить список без конкатенации?

Ответ: Расширение списка list_1 списком list_2 `list.extend(_iterable_)`

```
list_1 = ['dog', 'cat']
list_2 = ['mouse', 'frog', 'cat']
list_1.extend(list_2)
print(list_1)
```

Результат: ['dog', 'cat', 'mouse', 'frog', 'cat']

Вопрос 42. Как добавить элемент в список в конкретное место под конкретным индексом?

Ответ: Добавление элемента в список в конкретное место, под конкретным индексом
`list.insert(_i_, _x_)`

```
list_1 = ['John', 2.5, 'Cary', 'Ben', 4.2]
list_1.insert(3, 4.1)
print(list_1)
```

Результат: ['John', 2.5, 'Cary', 4.1, 'Ben', 4.2]

Вопрос 43. Как удалить элемент из списка с конкретным значением элемента?

Ответ: Удаление элемента из списка с конкретным значением `list.remove(_x_)`

Удаляет только 1 элемент, который первый нашел метод:

```
list_1 = ['Fred', 'Wilma', 'Barney', 'Dino', 'Fred', 'Wilma']
list_1.remove('Wilma')
print(list_1)
```

Результат: ['Fred', 'Barney', 'Dino', 'Fred', 'Wilma']

Вопрос 44. Что будет если при удалении элемента его нет в списке?

Ответ: `ValueError: list.remove(x): x not in list`

Вопрос 45. Как удалить элемент из списка по значению его индекса?

Ответ: Удаление элемента из списка со значением его индекса

`list.pop(_i_)` и возвращение удаленного элемента

```
list_1 = ['Fred', 'Wilma', 'Barney', 'Mr. Rock']
who = list_1.pop(3)
```

```
print(list_1)
print(who)
```

Результат:

['Fred', 'Wilma', 'Barney']

Mr. Rock

Если `list.pop()` записать без аргумента, то происходит удаление последнего элемента списка и возвращает элемент, который был удален

Вопрос 46. Как произвести поиск и возврат индекса по значению элемента?

Ответ: `list.index`(_x_[, _start_[, _end_]])

Например,

```
list_1 = ['Fred', 'Wilma', 'Barney', 'Mr. Rock']
who = list_1.index('Barney')
print(who)
```

Результат: 2

В методе есть значения start и end - они означают в пределах, каких значений мы ищем индексы.

Если ничего не нашли Python пишет ошибку ValueError.

Вопрос 47. Как произвести поиск и возврат одинаковых элементов списка?

Ответ: поиск и возврат количества одинаковых элементов в списке `list.count`(_x_)

```
fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple',
          'banana']
print(fruits.count('apple'))
```

Результат: 2 - слово 'apple' встречается 2 раза

Вопрос 48. Как производится сортировка элементов списка по возрастанию и убыванию?

Ответ: `list.sort`(_*__, _key=None_, _reverse=False_)

По возрастанию:

```
numbers = [5, 6, 2, 1, 8, 3]
numbers.sort()
print(numbers)
```

Результат: [1, 2, 3, 5, 6, 8]

По убыванию:

```
numbers = [5, 6, 2, 1, 8, 3]
numbers.sort(reverse=True)
print(numbers)
```

Результат: [8, 6, 5, 3, 2, 1]

Также используется универсальную функцию `sorted()`, но для этого нужна еще одна переменная:

```
numbers = [5, 6, 2, 1, 8, 3]
numbers_1 = sorted(numbers, reverse=False)
print(numbers_1)
```

Результат: [1, 2, 3, 5, 6, 8]

Надо помнить, что `sort()` - возвращает тот же самый список, а `sorted()` - другой

Вопрос 49. Как производится инверсия элементов списка?

Ответ: Инверсия элементов списка (наоборот) `'list.reverse'()`

```
numbers = [5, 6, 2, 1, 8, 3]
numbers.reverse()
print(numbers)
```

Результат: [3, 8, 1, 2, 6, 5]

Вопрос 50. Как скопировать список в другую ячейку памяти?

Ответ: Копирование списка, т.е. возврат такого же точно списка, но в другой ячейки памяти `'list.copy'()`

```
numbers = [5, 6, 2, 1, 8, 3]
copy = numbers.copy()
print(id(numbers))
print(id(copy))
```

Результат:

1562794928768 - ячейка памяти для списка numbers

1562794928320 - ячейка памяти для списка copy

Вместо функции `list.copy()` можно использовать срез

```
copy = a[:]
print(copy)
```

Вопрос 51. Как удалить все элементы списка?

Ответ: Удаление всех элементов списка `'list.clear'()`

```
numbers = [5, 6, 2, 1, 8, 3]
numbers.clear()
print(numbers)
```

Результат: []

Вопрос 52. Как из списка создать строку?

Ответ: Создание строки из списка `'str.join()'`

```
count = ['one', 'two', 'three']
words = '-'.join(count)
print(words)
```

Результат: one-two-three

Вопрос 53. Как сделать последовательный вызов всех элементов списка?

Ответ: с помощью оператора *

```
data = [1, 'fusk', 3.2, 'getting old']
print(*data)
```

Результат: 1 fusk 3.2 getting old

Вопрос 54. Как с помощью оператора * можно объединить 2 списка?

Ответ:

```
a = [1, 2, 3]
b = [*a, 4, 5, 6]
print(b)
```

Результат: [1, 2, 3, 4, 5, 6]

Вопрос 55. Что такое вложенные списки?

Ответ: это когда один список вложен в другой

LiveCoding: приведите пример вложенного списка:

```
img = [[1, 7, 6, 11, 3], [1, 7, 6, 11, 3], [1, 7, 6, 11, 3], [1,
7, 6, 11, 3], [1, 7, 6, 11, 3]]
```

Вопрос 56. Как происходит обращение к элементу вложенного списка?

Ответ: img[0][3]

Вопрос 57. Что такое кортеж?

Ответ: Кортежи tuple() - это НЕИЗМЕНЯЕМЫЕ итерируемая коллекция. Т.е. изменить никакой элемент невозможно, но обратиться к нему по индексу можно.

LiveCoding: дан список кортежей lt = [('Geeks', 2), ('For', 4), ('geek', '6')]. Преобразовать его в список ['Geeks', 2, 'For', 4, 'geek', '6']

Ответ:

```
lt = [('Geeks', 2), ('For', 4), ('geek', '6')]
out = [item for t in lt for item in t]
print(out)
```

Вопрос 58. В чем отличие списка от кортежа?

Ответ: 1) список – изменяемая коллекция, а кортеж – неизменяемая 2) список не может быть ключом словаря, а кортеж может 3) кортежи занимают меньший объем памяти в компьютере и быстрее работают. 4) кортежи защищены от изменений

Вопрос 59. Какая особенность записи кортежей, если мы хотим создать кортеж с одним элементом – числом 1?

Ответ: записываются в круглые скобки, если элемент 1 – обязательно после него запятая (1,)

Вопрос 60. Как объявляется кортеж?

Ответ: можно в круглых скобках, а можно через запятую

```
my_first_tuple = 1, 2, 3, 'Amazing', 1.72
print(my_first_tuple)
```

Результат: (1, 2, 3, 'Amazing', 1.72)

Вопрос 61. Как производится распаковка кортежа

Ответ:

```
my_second_tuple = ('Fred', 3.14, None, 17)
name, fl, is_none, integers = my_second_tuple
```

Вопрос 62. Как узнать число элементов в кортеже?

Ответ: с помощью функции len()

Вопрос 63. Как обратиться к элементу кортежа по индексу?

Ответ:

```
my_fourth_tuple = ('Banana', 'Apple', 'Mango')
print(my_fourth_tuple[2])
```

Результат: Mango

Вопрос 64. Как удалить элемент кортежа?

Ответ: элемент кортежа удалить нельзя, но можно удалить сам кортеж полностью

```
my_fifth_tuple = (4, 5, 6)
del my_fifth_tuple
print(my_fifth_tuple)
```

Результат: NameError: name 'my_fifth_tuple' is not defined

Вопрос 65. Можно ли объединить 2 кортежа в 3ий?

Ответ: да, можно

```
a = (2, 3, 4)
b = (1, )
print(a + b)
(2, 3, 4, 1)
```

Вопрос 66. Какие типы данных можно превратить в кортеж и как?

Ответ:

- Списки

```
a = tuple([1, 2, 3])
(1, 2, 3)
```

- Строки

```
b = "hello"
("h", "e", "l", "l", "o")
```

Вопрос 67. Допустим, что в кортеже один из элементов – список. Можно ли в этот элемент кортежа добавить элемент?

Ответ: да, можно

```
a = (True, 9, [1, 2, 3, 4], 8.2, {"word": "слово"})
a[2].append(666)
print(a)

(True, 9, [1, 2, 3, 4, 666], 8.2, {'word': 'слово'})
```

Вопрос 68. Как посчитать сколько элементов в кортеже с определенным значением

Ответ: с помощью метода count

```
a = (True, 9, [1, 2, 3, 4], 8.2, {"word": "слово"}, 9, 9.86)
print(a.count(9))
2
```

Вопрос 69. Как найти индекс первого элемента со значением в кортеже?

Ответ: с помощью функции index

```
a = (True, 9, [1, 2, 3, 4], 8.2, {"word": "слово"}, 9, 9.86)
print(a.index(9))
1
```

Вопрос 70. Что такое range()?

Ответ: Диапазон range() - это не тип данных, а встроенная функция, в которой хранится последовательность целых чисел.

Вопрос 71. Как сделать список с помощью диапазона range()?

Ответ:

```
a = list(range(1, 20, 2))
print(a)
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

Вопрос 72. Как задать последовательность диапазона range() в обратном порядке?

Ответ: изменить шаг range() на отрицательный

```
a = list(range(0, -5, -1))
print(a)
[0, -1, -2, -3, -4]
```

Вопрос 73. Как использовать * (оператор упаковки и распаковки) для диапазона range?

Ответ:

```
d = -5, 5
print(list(range(*d)))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
```

Вопрос 74. Что такое строковый тип данных?

Ответ: Строковый тип данных str() – это неизменяемая итерируемая последовательность, только символов. Каждый символ имеет точно такой же индекс, как, например, элементы списка.

Вопрос 75. Что такое f-строки?

Ответ: это строковый тип данных, а котором можно прописывать переменные с помощью фигурных скобок

Вопрос 76. Что такое конкатенация строк?

Ответ: это сложение двух строк, результатом является третья строка

```
s1 = "I love"
s2 = " you"
s3 = s1 + s2
print(s3)
I love you
```

Вопрос 77. Как можно сделать дублирование строк?

Ответ:

```
s1 = " xa" * 5
print(s1)
xa xa xa xa xa
```

Вопрос 78. Как определить количество символов в строке?

Ответ

```
s1 = "I do not know"
print(len(s1))
13
```

Вопрос 79. Как определить, что одна строка входит в другую без использования алгоритмов (типа КМП)?

Ответ: Оператор in для строк

```
s1 = "ab"
s2 = "abracadabra"
print(s1 in s2)
True
```

Вопрос 80. Как производится сравнение строк?

Ответ: Сравнение строк (чувствительность к регистру букв)

```
s1 = "hello"
print(s1 == "Hello")
False
```

Вопрос 81. Как узнать числовое значение символа по таблице ASCII?

Ответ: с помощью функции ord()

```
print(ord("Z"))
90
```

Вопрос 82. Как сделать срез строк?

Ответ:

```
s = "it does not matter"
print(s[12:18])
matter
```

Вопрос 83. Как в срезе задать шаг перебора символов?

Ответ:

```
s = "it does not matter"
print(s[12:18:2])
mte
```

Можно указать и так [5::3] - это означает, что от 5 символа до конца через 3 символа

Вопрос 84. Как инвертировать строку с помощью среза?

Ответ:

```
s = "it does not matter"
print(s[::-1])
rettam ton seod ti
```

Вопрос 85. Как обратить строку в множество set()?

Ответ:

```
name = 'Василий'
print(set(name))
Результат: {'В', 'с', 'й', 'и', 'а', 'л'}
```

Надо заметить, что в этом множестве нет повторяющихся символов - их автоматически убирает множество set{}.

Вопрос 86. Как сделать обращение к элементу строки по индексу?

Ответ:

```
animal = 'butterfly'
print(animal[5])
```

Результат: r

Индекс может быть и отрицательным:

```
animal = 'butterfly'
print(animal[-7])
```

Результат: t

Вопрос 87. Как произвести разделение строки на слова или другие фрагменты текста и сохранить полученное в список list?

Ответ: с помощью метода str.split(_sep=None, _maxsplit=- 1_)

```
numbers = '1, 2, 3, four, 5'
list_numbers = numbers.split(', ')
print(list_numbers)
```

Результат: ['1', '2', '3', 'four', '5']

Всегда требуется новая переменная, потому что список - это отдельный объект

если использовать просто list.split() - без аргументов в скобках, то будет произведено разбиение по пробелам

Вопрос 88. Как создать строки из списка?

Ответ: методом str.join()

```
count = ['one', 'two', 'three']
words = '-'.join(count)
print(words)
```

Результат: one-two-three

Вопрос 89. Как сделать перевод всех символов из ВЕРХНЕГО РЕГИСТРА в нижний регистр?

Ответ: методом str.lower()

```
status = 'ALL ALRIGHT'
st_low = status.lower()
print(st_low)
```

Результат: all alright

Стоит заметить, что этот метод создает копию строки!

Вопрос 90. Как сделать перевод всех символов из нижнего регистра в ВЕРХНИЙ РЕГИСТР?

Ответ: с помощью метода str.upper()

```
status = 'nothing else'
st_up = status.upper()
print(st_up)
```

Результат: NOTHING ELSE

Вопрос 91. Как сделать первый символ в верхнем регистре, а остальные в нижнем?

Ответ: методом str.capitalize()

```
status = 'What Do YOU thinK!'
st_up = status.capitalize()
print(st_up)
```

Результат: What do you think!

Вопрос 92. Как сделать, чтобы первый символ каждого нового слова переведен в верхний регистр, остальные в нижний?

Ответ: методом str.title()

```
status = 'diCk cHaNEY'
st_up = status.title()
print(st_up)
```

Результат: Dick Chaney

Вопрос 93. Как произвести замену слова?

Ответ: методом str.replace(src, new)

```
phrase = "I'm nothing"
true_phrase = phrase.replace('nothing', 'the best')
print(true_phrase)
```

Результат: I'm the best

В этом методе после new через запятую можно написать максимальное кол-во замен:

```
s = "My best friend is Python!"
print((s.replace(" ", "'", 1)).replace(" ", "\\ ")))
```

Вопрос 94. Как удалить из строки ненужные элементы?

Ответ: методом `str.strip()`

```
rude = 'ERROR404: are you an idiot?'
rude_1 = rude.strip('ERROR404: ')
print(rude_1)
```

Результат: are you an idiot?

Вопрос 95. Как вернуть число повторений подстроки в строке?

Ответ: методом `str.count()`

```
s = "abrakadabra"
print(s.count("ra"))
2
```

Через запятую можно указать откуда начинать поиск `str.count("some", 3, 10)`

Вопрос 96. Как проверить из чего состоит строка: только ли из букв или только ли из цифр?

Ответ:

Проверка из чего состоит строка методом ``str.isalpha()``

Если только буквы в строке `True`

Если не только буквы в строке - `False`

``Метод str.isdigit()``

Если только цифры - `True`

Если что-то есть помимо цифр – `False`

Вопрос 97. Перечислите специальные символы в строках?

Ответ: ``\n`` - перевод строки на другую строчку (`print` автоматически делает перенос строки, видя этот символ)

``\t`` - делает отступ строки в виде табуляции

``\\`` - экранирование, символы с двойным назначением. Приписывается при написании пути к файлу или URL, чтобы не было ``\t`` или ``\n``

Правильное написание:

```
`s = "Марка вина \"Ягодка\""
```

или использовать одинарные кавычки

Вопрос 98. Что такое r-строки?

Ответ:

r-строки - это raw-строки - строки, где все символы воспринимаются также, как и написано

```
s = "D:\\comples\\python.exe"
print(s)

s = r"D:\\comples\\python.exe"
print(s)
```

```
D:\comples\python.exe - без raw-строк
D:\\comples\\python.exe - с raw-строкой
```

Вопрос 99. Что такое множества?

Ответ: Множества `set{}` - это изменяемая неупорядоченная коллекция.

Вопрос 100. Можно ли изменить отдельный элемент в множестве?

Ответ: Изменить отдельный элемент в множестве нельзя.

Вопрос 101. Что может быть элементами множества?

Ответ: Элементами множества могут быть только неизменяемые объекты: числа, строки, кортежи, булевы значения, `NoneType`

Вопрос 102. Возможно ли наличие двух и более одинаковых элементов в множестве?

Ответ: Во множестве не может быть двух одинаковых элементов.

Вопрос 103. Что будет, если одинаковые элементы попадают в множество?

Ответ: Если одинаковые элементы попадают во множество, то первый удаляется, а второй сохраняется в нем.

Вопрос 104. Как из списка можно получить множество?

Ответ: Множество можно получить из списка, тогда в полученном множестве удалятся все повторяющиеся элементы

```
letters = ['a', 'b', 'c', 'b']
uniq_letters = set(letters)
print(uniq_letters)
```

Результат: `{'a', 'b', 'c'}`

Вопрос 105. Как можно добавить элемент в множество?

Ответ: добавление элемента во множество `set.add()`:

```
fruits = {'banana', 'mango'}
fruits.add('apple')
print(fruits)
```

Результат: {'mango', 'apple', 'banana'}

Вопрос 106. Какие операции можно производить над множествами?

Ответ Объединение двух множеств `set.union()`, Поиск различий в двух множествах `set.difference()`, Поиск одинаковых элементов двух множеств `set.intersection()`, Удалить элемент множества методом `set.remove()`, Удаление всех элементов множества `set.clear()`:

Вопрос 107. Что такое словари в Python и из чего они состоят?

Ответ: Словарь `dict{}` - это упорядоченная (с версии Python 3.7) коллекция из пар элементов, первый элемент называется `key` (ключ), второй элемент называется `value` (значение).

Вопрос 108. Что может быть ключом в словаре?

Ответ: Ключом может быть любой неизменяемый тип данных: `int()` - целые числа, `float()` - числа с плавающей точкой (запятой), `complex()` - комплексные числа, `str()` - строковый тип данных, `tuple()` – кортежи, `bool` - булевый тип

Вопрос 109. Что может быть значением в словаре?

Ответ: Значением может быть любой объект

Вопрос 110. Как получить значение ключа словаря?

Ответ:

```
book = {'First': 'Harry Potter', 'Second': 'Lord of The Rings'}
print(book["First"])
```

Результат: Harry Potter

Это не единственный способ - есть еще метод `get()` - см. пункт 10. Но отличие в том, что если ключ здесь не найден - то вернется ошибка, а в `get` – `None`

Вопрос 111. Как можно преобразовать ключи словаря в список?

Ответ:

```
book = {'First': 'Harry Potter', 'Second': 'Lord of The Rings'}
not_book = list(book)
print(not_book)
```

Результат: ['First', 'Second']

Вопрос 112. Как можно преобразовать ключи словаря во множество?

Ответ:

```
book = {'First': 'Harry Potter', 'Second': 'Lord of The Rings'}
not_book = set(book)
print(not_book)
```

Результат: {'First', 'Second'}

Вопрос 113. Как можно создать словарь из списка кортежей?

Ответ:

```
movies = [('Матрица', 4.7), ('Трон', 3.8)]
movies_dict = dict(movies)
print(movies_dict)
```

Результат: {'Матрица': 4.7, 'Трон': 3.8}

Вопрос 114. Как можно создать словарь с помощью непосредственного присвоения аргументов?

Ответ:

```
movies_dict = dict(Матрица = 4.7, Трон = 3.8, Хакеры = 4.3)
print(movies_dict)
```

Результаты: {'Матрица': 4.7, 'Трон': 3.8, 'Хакеры': 4.3}

Вопрос 115. Назовите еще один способ создания словаря?

Ответ: Создание словаря методом dict.fromkeys()

```
movies = dict.fromkeys(['Матрица', 'Хакеры', 'Трон', 'Кибер'])
print(movies)
```

Результат: {'Матрица': None, 'Хакеры': None, 'Трон': None, 'Кибер': None}

Т.к. после скобки] нет запятой и значения, то значения принимаются None

Вопрос 116. Как проверить есть ли ключ в словаре?

Ответ: Проверить, есть ли ключ в словаре методом in

```
capitals = {'Russia': 'Moscow', 'Norway': 'Oslo'}
print('Russia' in capitals)
print('Sweden' in capitals)
```

Результат:

True

False

Вопрос 117. Как добавить новый элемент в словарь?

Ответ:

```
capitals = {'Russia': 'Moscow', 'Norway': 'Oslo'}  
capitals['Finland'] = 'Helsinki'  
print(capitals)
```

Результат:

```
{'Russia': 'Moscow', 'Norway': 'Oslo', 'Finland': 'Helsinki'}
```

Вопрос 118. Как можно объединить 2 словаря?

Ответ:

```
letters1 = {'A': 'a', 'B': 'b'}  
letters2 = {'C': 'c', 'D': 'd'}  
letters1.update(letters2)  
print(letters1)
```

Результат: {'A': 'a', 'B': 'b', 'C': 'c', 'D': 'd'}

Также возможен такой синтаксис:

```
letters1 = {'A': 'a', 'B': 'b'}  
letters2 = {'C': 'c', 'D': 'd'}  
c = {**letters1, **letters2}  
print(c)
```

`{'A': 'a', 'B': 'b', 'C': 'c', 'D': 'd'}`

Вопрос 119. Как удалить элемент из словаря?

Ответ:

```
pets = {'Fiendly': 'dogs', 'Neitral': 'sheep'}  
del pets['Neitral']  
print(pets)
```

Результат: {'Fiendly': 'dogs'}

Вопрос 120. Как удалить элемент словаря и вернуть значение удаляемого элемента?

Ответ: dict.pop()

```
words = {'Letter': 'буква', 'Dog': 'собака'}  
deleted = words.pop('Dog')  
print(words)  
print(deleted)
```

Результат:

```
{'Letter': 'буква'}
```

Собака

Вопрос 121. Как получить все ключи и все значения словаря?

Ответ:

```
family_guy = {'Peter': 'male', 'Lois': 'woman', 'Brian': 'dog'}  
print(family_guy.keys())  
print(family_guy.values())
```

Результат:

dict_keys(['Peter', 'Lois', 'Brian'])

dict_values(['male', 'woman', 'dog'])

Вопрос 122. Как перебрать по ключам словарь и как по значениям?

Ответ:

```
family_guy = {'Peter': 'male', 'Lois': 'woman', 'Brian': 'dog'}  
for unit in family_guy:  
    print(unit)
```

Результат:

Peter

Lois

Brian

Перебор по значениям:

```
family_guy = {'Peter': 'male', 'Lois': 'woman', 'Brian': 'dog'}  
for unit in family_guy.values():  
    print(unit)
```

Результат:

male

woman

dog

Вопрос 123. Как получить из словаря пары ключ-значение и как их перебрать?

Ответ:

```
family_guy = {'Peter': 'male', 'Lois': 'woman', 'Brian': 'dog'}  
print(family_guy.items())
```

Результат:

dict_items([('Peter', 'male'), ('Lois', 'woman'), ('Brian', 'dog')])

Далее можно циклом for распаковать кортежи:

```
family_guy = {'Peter': 'male', 'Lois': 'woman', 'Brian': 'dog'}  
for name, gender in family_guy.items():  
    print(f'{name} is a {gender}')
```

Результат:

Peter is a male

Lois is a woman

Brian is a dog

Вопрос 124. Как удалить все элементы словаря?

Ответ: dict.clear()

Вопрос 125. Как сделать копию словаря?

Ответ: dict.copy()

Вопрос 126. Как вернуть значение по заданному ключу и если нет ключа он его создаст в словаре это ключ

Ответ:

```
dc = {1: 'good', 2: 'not good'}
dc.setdefault(3, 'very bad')
print(dc)
a = dc.setdefault(2, 'very bad')
print(a)
{1: 'good', 2: 'not good', 3: 'very bad'}
not good
```

Вопрос 127. Что такое тернарный условный оператор?

Ответ это условный оператор if – else, записанный в одну строку

```
a = 12
b = 7
res = a if a > b else b
print(res)
12
```

Вопрос 128. Примените тернарный условный оператор для строк?

Ответ:

```
a = 12
print("a - " + ("четное" if a % 2 == 0 else "нечетное"))
Результат: a - четное
```

Круглые скобки в этом случае необходимы.

Вопрос 129. Какие циклы бывают в Python?

Ответ: бывают for – цикл со счетчиком, и цикл с условием while

LiveCoding: продемонстрируйте на любых примерах работу этих циклов, в том числе и совместно, а также работу вложенных циклов

Вопрос 130. Какую роль в цикле while играет ключевое слово else?

Ответ: После тела цикла можно прописать необязательный блок операторов else и этот блок операторов будет выполняться после штатного завершения цикла while

Штатное завершение - это когда условие цикла while стало равно false

Прерывание цикла по break является нештатным,

Вопрос 131. Что такое итератор в Python. Приведите примеры?

Ответ: В Python существует функция iter(). Каждый элемент из этого итератора можно получить с помощью функции next(). Доступ к элементу через итератор и по индексу - это совершенно разные вещи.

Для последнего элемента возникает StopIteration

```
lst = list(map(str, input().split()))
it = iter(lst)
print(next(it))
print(next(it))
```

Вопрос 132. Для чего нужны iter() и next()?

Ответ: Если нам нужно постоянно перебирать итерируемые объекты самых разных типов, то единственный универсальный и безопасный метод - использовать итератор

Оператор for основан на iter() и next()

Вопрос 133. Как одновременно при переборе элементов списка, кортежа сразу брать индекс и значение элемента. Приведите пример?

Ответ: Чтобы из любого итерируемого объекта сразу брать и индекс и значение существует функция enumerate.

```
digs = [4, 3, 100, -53, -30, 1, 34, -8]
for i, d in enumerate(digs):
    if 10 <= abs(d) <= 99:
        digs[i] = 0
print(digs)
```

Вопрос 134. Как можно сгенерировать список?

Ответ: Если цикл for можно заменить на list comprehension - то лучше так делать, т.к. list comprehension намного быстрее и компактнее чистого цикла for.

```
N = 6
a = [x ** 2 for x in range(N)]
print(a)
[0, 1, 4, 9, 16, 25]
```

Вопрос 135. Приведите пример использования тернарного оператора при генерации списка?

Ответ:


```
d = [-3, 0, 6, -12, 2, 12, 0, -2, 1, 0, 5]
a = ["четное" if x % 2 == 0 else "нечетное" for x in d]`
print(a)
['нечетное', 'четное', 'четное', 'четное', 'четное', 'четное', 'четное', 'четное', 'нечетное', 'четное',
'нечетное']
```

Вопрос 136. Покажите на любом примере генератор словаря?

Ответ:

```
movies = ['Матрица', 'Хакеры', 'Трон']
category = 'Фантастика в IT'
movies_info = {movie:category for movie in movies}
print(movies_info)
Результат: {'Матрица': 'Фантастика в IT', 'Хакеры': 'Фантастика в IT', 'Трон': 'Фантастика в IT'}
```

Вопрос 137. Как сделать обратное преобразование ключей в значения, а значений в ключи?

Ответ:

```
d = {'отлично':5, 'хорошо': 4, 'удовлт': 3, 'неуд':2}
a = {value: key for key, value in d.items()}
print(a)
{5: 'отлично', 4: 'хорошо', 3: 'удовлт', 2: 'неуд'}
```

Вопрос 138. Как сделать генератор множеств?

Ответ:

```
d = [1, 2, 3, 2, '5', '3', 3, 4]
a = {int(x) for x in d}
print(a)
{1, 2, 3, 4, 5}
```

Вопрос 139. Что такое функция в Python и из какой части программы можно вызвать функцию?

Ответ: Функция - это именованный блок кода, выполняющий определенную задачу. Функцию можно вызвать из любой части программы.

Вопрос 140. Какие типы аргументов могут быть переданы в параметры функции?

Ответ: Аргументы, которые идут в параметры функции, могут быть позиционными или именованными.

Вопрос 141. Приведите примеры позиционных аргументов?

Ответ:

```
def these(a,b,c):
    return f'Positional - {a}, {b}, {c}'
print(these(1,2,3))
```

Результат: Positional - 1, 2, 3

Вопрос 142. Приведите пример именованных аргументов?

Ответ:

```
def these(a,b,c):  
    return f'Named - {a}, {b}, {c}'  
print(these(a=3, b=4, c='V'))
```

Результат: Named - 3, 4, V

Вопрос 143. Какие могут быть параметры у функции?

Ответ:

ФАКТИЧЕСКИЕ ПАРАМЕТРЫ - это параметры функции, которые не имеют начально-заданных значений

```
def these(a,b,c)
```

ФОРМАЛЬНЫЕ ПАРАМЕТРЫ - это параметры функции, которые сразу имеют определенное значение

```
def these(verbose=True)
```

Вопрос 144. Как задать в функции параметры по-умолчанию?

Ответ:

```
def print_these(a=None,b=None,c=None):  
    print(a, "is stored in a")  
    print(b, "is stored in b")  
    print(c, "is stored in c")  
print_these(c=3, a=1)
```

Результат:

1 is stored in a

None is stored in b

3 is stored in c

Вопрос 145. Что делать, если при объявлении функции произвольное число фактических и формальных параметров?

Ответ: *args и **kwargs используются при объявлении функции с произвольным числом фактических и формальных параметров. *args формирует на входе список args, **kwargs формирует на входе словарь kwargs

Вопрос 146. Что такое рекурсия?

Ответ: Рекурсивная функция - это функция, которая вызывает саму себя.

Вопрос 147. Для чего нужна рекурсия?

Ответ: Рекурсия решает задачу, вызывая в теле функции саму себя, что представляет собой более простые подзадачи.

Вопрос 148. Что требуется для того, чтобы правильно работала рекурсия?

Ответ: Для решения рекурсии требуется разрешить 2 случая:

1. Рекуррентный случай - как мы будем решать задачу, уменьшая ее сложность. Рекуррентный случай может быть, как один, так и несколько
 2. Крайний случай - случай, когда рекурсия останавливается.
-

Вопрос 149. Приведите пример рекурсивной функции?

Ответ:

```
def fact(n):  
    if n <= 1:  
        return 1  
    return fact(n - 1) * n  
  
print(fact(6))
```

Результат: 720

Вопрос 150. Какое отличие рекурсии от цикла?

Ответ: Отличие рекурсии от цикла: цикл не упрощает задачу и не разбивает её на решения подзадач в отличие от рекурсии. Но цикл можно свести к рекурсии. В функциональном программировании циклы создаются при помощи рекурсии

Вопрос 151. Каков порядок решения задач на рекурсию?

- Ответ:
1. Проверить, не является ли этот случай крайним
 2. Прописать условие крайнего случая
 3. Прописать аргументы которые будут передаваться в рекуррентную функцию
 4. Прописать рекуррентную функцию в теле функции
 5. Вызвать функцию в глобальной области
-

Вопрос 152. Опишите процесс работы рекурсии в памяти компьютера. Что может произойти, когда не задан крайний случай?

Ответ: Когда вызывается какая-нибудь функция она автоматически помещается в стек вызова функции, в котором хранится порядок вызова функции. В этот стек при вызове помещается все новая и новая функции, которая была вызвана из начальной. Но стек не бесконечный и на определенной итерации, идет переполнение стека.

Вопрос 153. Что такое глубина рекурсии?

Ответ: Глубина рекурсии - это есть количество самовывозов функции. Глубина постепенно увеличивается, и мы можем увидеть ту часть кода до погружения и до восстановления.

Вопрос 154. Что такое лямбда-функция. Приведите пример?

Ответ: Лямбда-функция - это анонимная функция. f - объект принимает и хранит результат выражения (expression)

lambda - ключевое слово

a: - аргумент

a * a - выражение, записанное в одну строку

Пример,

```
f = lambda a,b: a + b
print(f(4,6))
```

Результат: 10

Вопрос 155. Нужен ли return в лямбда-функции?

Ответ: Лямбда-функция автоматически возвращает результат выражения, никакого return не нужно

Вопрос 156. Может ли быть в выражении лямбда-функции тернарный оператор?

Ответ: может

Вопрос 157. В чем отличие лямбда-функции от обычной функции?

Ответ: Lambda-функция может быть записана как элемент любой конструкции языка Python. Например, она может быть одним из элементов списка:

```
lst = [4, True, 8, 9.65, None, lambda: print("lambda"), False]
for i in lst:
    print(i, end=" ")

4 True 8 9.65 None <function <lambda> at 0x00000253BE9C5798>
False
```

Чтобы выполнить lambda-функцию её надо вызвать:

```
lst[5]()
```

Результат: lambda

Вопрос 158. Какие ограничения есть у лямбда-функций?

Ответ: можно прописывать только одно выражение и не больше, внутри lambda-функций **нельзя использовать оператор присваивания = и инкрементирование -=, +=, *=, /=**

Вопрос 159. Что такое глобальные и локальные переменные, приведите примеры?

Ответ: Локальная переменная - это такая переменная, которая существует только внутри функции, и за пределами этой функции мы не можем к ним обращаться

Глобальная переменная - это переменная, к которой можно обращаться в любом месте программы, в том числе и в функции

```
N = 100 #Глобальная переменная

def get_a(a):
    N=10 # Независимая локальная переменная
    return a + N

print(get_a(0)) # Результат суммы локальной переменной
print(N) # Глобальная не изменяется
```

Вопрос 160. Объясните суть ключевого слова global. Приведите пример?

Ответ: Когда внутри функции мы хотим работать с глобальной переменной, надо прописать ключевое слово global. Это означает, что переменная берется из глобальной области, а не создается новая в локальной области

```
1  N = 100 #Глобальная переменная
2
3  def get_a(a):
4      global N # Внутри функции создается новая глобальная переменная
5      N=10 # И ей присваивается значение 10
6      return a + N
7
8  print(get_a(20)) # = 30
9  print(N) # =10
10
```

Вопрос 161. Объясните суть ключевого слова nonlocal?

Ответ:

```

x = 10 # Глобальная переменная всей программы
def exterior():
    x = 5 # Локальная переменная функции
    def interior():
        nonlocal x # Здесь x внутри interior НЕ СОЗДАЕТСЯ
        x = 2 # Она берется из exterior и ей присваивается 2
        print(f'interior={x}')

    interior()
    print(f'exterior={x}')
exterior()
print(f'global={x}')

```

nonlocal - можно писать только в том пространстве имен, которое ссылается на другое локальное пространство

Вопрос 162. Дайте определение замыканиям в Python?

Ответ: ЗАМЫКАНИЕ - это способность вложенной функции запоминать локальное состояние контекстной области внешней (объемлющей) функции

Вопрос 163. Объясните механизм замыкания с примерами?

Ответ:

1. Мы знаем, что в существующей функции можно объявить еще одну функцию, которая будет вложенной.

```

1  def say_name(name):
2      def say_goodbye():
3          print("Don't say goodbye ", name)
4
5      return say_goodbye()
6
7  f = say_name("Ivan")
8  f

```

2. В существующей функции сделаем возвращение внутренней функции

```

def say_name(name):
    def say_goodbye():
        print("Don't say goodbye ", name)

    return say_goodbye()

f = say_name("Ivan")
f

```

3. Сохраняем ссылку на существующую функцию и вводим аргумент:

```

1 def say_name(name):
2     def say_goodbye():
3         print("Don't say goodbye ", name)
4
5     return say_goodbye()
6
7 f = say_name("Ivan")
8

```

4. Вызвать внутреннюю функцию через ссылку f

```

1 def say_name(name):
2     def say_goodbye():
3         print("Don't say goodbye ", name)
4
5     return say_goodbye()
6

```

5. Результат будет такой

```

def say_name(name):
    def say_goodbye():
        print("Don't say goodbye ", name)

```

D:\stepik\new\venv\Scripts\python.exe
Don't say goodbye Ivan

По идее, существующая функция say_name должна была закончиться на операторе return. Но в return она вызывает результат внутренней функции, а вложенная функция имеет способность запоминать значение параметра внешней функции

Ключевое здесь - это переменная f, которая даёт ссылку на внутреннее локальное окружение, то это окружение продолжает существовать. Оно продолжает существовать, а не удаляется автоматически сборщиком мусора. Вместе с локальным окружением функции say_goodbye() продолжают существовать все внешние окружения, которые с ним связаны.

У каждого локального окружения есть неявная/скрытая ссылка на внешнее окружение.

say_goodbye неявно ссылается на say_name

say_name неявно ссылается на глобальную область

И все эти окружения держатся, они не пропадают, пока существует эта ссылка f

ЗАМЫКАНИЕ - это цепочка переменная f (принадлежит глобальной области видимости), переменная f ссылается на локальное окружение функции say_goodbye(), а say_goodbye ссылается на внешнее окружение функции say_name, а say_name на глобальную область. Если мы передаем разные аргументы, то будут создаваться совершенно разные локальные окружения, но место в памяти ПК, где хранится функция будет та же самая:

Вопрос 164. Дайте определение декоратора в Python?

Ответ: Декоратор - это обёртка-замыкание, которая изменяет поведение декорируемой функции и расширяет её функционал. Сама же декорируемая функция при этом не модифицируется.

Вопрос 165. Напишите в общем виде функцию декоратор и как она работает?

Ответ:

```
1 def func_decorator(func): # Название декоратора, параметр - декорируемая функция
2     def wrapper(*args, **kwargs):
3         print("-actions-") # Выполняет команды до вызова декорируемой функции
4         result = func(*args, **kwargs) # Вызов декорируемой функции
5         print("-actions-") # Выполняет команды после вызова декорируемой функции
6         return result # Возврат результата декорируемой функции
7
8     return wrapper # Замыкание
9
10
11 def some_func(title, tag):
12     print(f"title = {title}, tag = {tag}")
13     return f"<{tag}>{title}</{tag}>"
14
15
16 f = func_decorator(some_func)
17
18 tagged = f("python", "h1")
19 print(tagged)
```

Результат этой выполнения этой функции будет такой:

```
↑ D:\stepik\new\venv\Scripts\python.exe D:\stepik
↓
-actions-
title = python, tag = h1
-actions-
<h1>python</h1>
```

У нас есть некая переменная f в глобальной области видимости, которая ссылается на внутреннюю функцию wrapper(), определенной во внешней функции func_decorator(func), а параметр func он ссылается на функцию some_func()

f вызывает wrapper(), wrapper() вызывает some_func(), НО дополнительно что-то делает перед её вызовом и после её вызова, тем самым расширяем её функционал.

Вопрос 166. Для чего нужны во wrapper args и kwargs?

Ответ: Они делают декоратор универсальным, у нас на вход может приходить все что угодно - args и kwargs принимают произвольное число фактических параметров args, и формальных параметров kwargs.

Вопрос 167. Для чего в декораторах иногда используются параметры?

Ответ: в декораторы можно передавать параметры для того, чтобы, например, увеличить точность вычислений. Например, производной Вычисление производной с заданной точностью - это задача декоратора, а сама функция, которая будет вычислять, например, синус по значению:

```
def df_decorator(dx=0.01):
    def func_decorator(func):
        def wrapper(x, *args, **kwargs):
            res = (func(x + dx, *args, **kwargs) - func(x, *args, **kwargs)) / dx
            return res
        return wrapper
    return func_decorator

sin_df = df_decorator(dx=0.001)(sin)

df = sin_df(math.pi/3)

print(df)
```

Вопрос 168. Каким символом может быть вызван декоратор?

Ответ: Можно сделать вызов с помощью @

```
@df_decorator(dx=0.000001)
def sin_df(x):
    return math.sin(x)

print(sin_df(math.pi/3))
```

Вопрос 169. Что такое @wraps и для чего он нужен?

Ответ: Когда у нас происходит декорирование функции с параметрами, впоследствии невозможно будет вызвать имя декорируемой функции и её описание (docstrings). Для решения этой проблемы служит декоратор @wraps:

```

import math
from functools import wraps

def df_decorator(dx=0.01):
    def func_decorator(func):
        @wraps(func)
        def wrapper(x, *args, **kwargs):
            res = (func(x + dx, *args, **kwargs) - func(x, *args, **kwargs)) / dx
            return res
        return wrapper
    return func_decorator

@df_decorator(dx=0.000001)
def sin_df(x):
    """Возвращает значение синуса по аргументу"""
    return math.sin(x)

print(sin_df.__name__)
print(sin_df.__doc__)

```

Теперь можно узнать имя и докстринг декорируемой функции:

```

D:\stepik\new\venv\Scripts\python.exe D:\st
sin_df
Возвращает значение синуса по аргументу

```

Вопрос 170. Что такое конструкция `if __name__ == __main__` и для чего она нужна?

Ответ: Потребность в коде этой конструкцией существует, когда есть несколько ру-файлов в одной директории (папке).

Дело в том, что когда мы импортируем пользовательские модули, они компилируются в байт-код и файл, где прописан импорт, выполняет его. Таким образом, подключаемые модули должны СОДЕРЖАТЬ ТОЛЬКО ЗНАЧЕНИЯ ПЕРЕМЕННЫХ, ФУНКЦИИ И КЛАССЫ, НО НЕ ИХ ВЫЗОВЫ.

В каждом модуле есть специальная переменная `__name__`. В зависимости от того, какой файл мы запускаем `__name__` меняет значение на `__main__`.

Но если мы импортируем какой-нибудь файл, и запускаем другой файл, куда он был импортирован, то `__name__` модуля так и остается `__name__` того модуля,

`if __name__ == __main__` нужна, чтобы команда исполнялась только при непосредственном запуске этого модуля.

Вопрос 171. Что такое пакеты в Python?

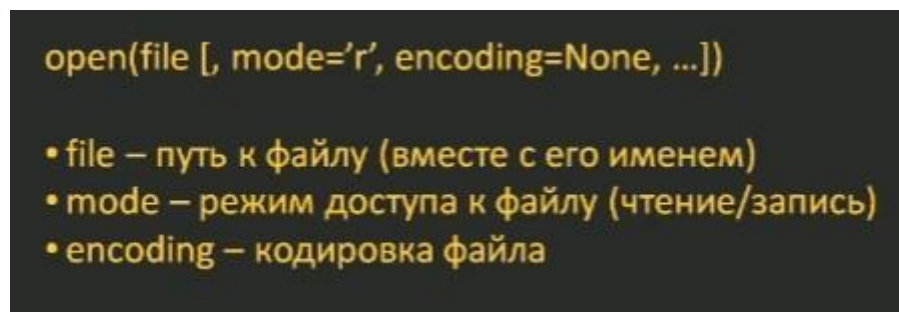
Ответ: Пакет (package) - это специальным образом организованный подкаталог с набором py-файлов, как правило, решающий сходные задачи. В нем есть `__init__.py` - это инициализатор пакета. Т.е. там прописывается, что будет импортировано из пакета при импорте в файле `main.py`.

Вопрос 172. Что такое файлы в Python?

Ответ: Файл в Python - это итерируемый объект, который хранится на внешнем носителе и состоит из строк и символов.

Вопрос 173. Какая основная функция при работе с файлами?

Ответ: Основная функция при работе с файлами - это функция `open()`



`mode` - режим доступа к файлу:

- `mode='r'` - открыть файл для чтения (стоит по умолчанию)
- `mode='w'` - открыть файл для записи. Создает новый файл, если он не существует. А если он существует - обрезает файл
- `mode='a'` - открывает файл для добавления к концу файла без его усечения. Создает новый файл, если он не существует

`encoding` - формат кодировки файла:

- `encoding='utf-8'` - открывает файл в UTF-8 кодировки

Вопрос 174. Расскажите о чтении информации из файла, о файловой позиции?

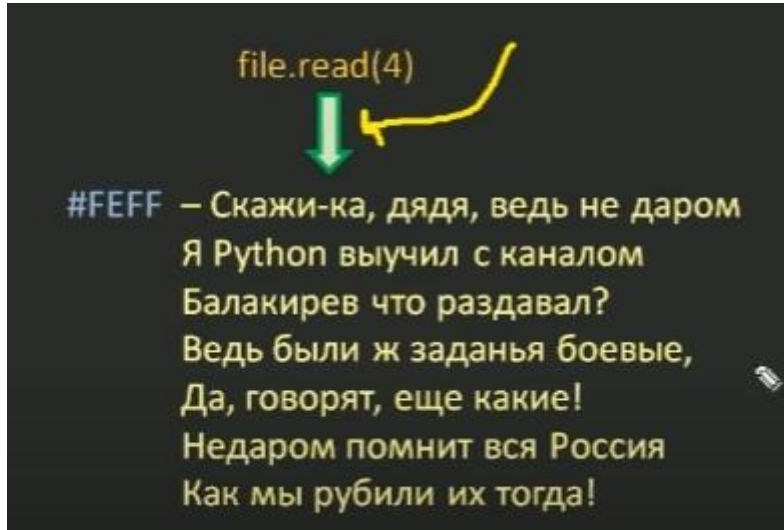
Ответ: ## Функция `read()`

Если в круглых скобках ничего не указано, то читается весь файл. Если указать в ней целочисленное значение, то прочитается несколько символов.

У нас отобразилось только 4 символа, а не 5. Это связано с тем, что в utf-8 присутствует невидимый символ

```
file = open("my_file.txt")  
  
print(file.read(5))
```

Зеленая стрелка - это файловая позиция, она указывает с какого места считывать данные из файла.



Вопрос 175. Что такое EOF?

Ответ: Когда мы доходим до самого конца файла появляется EOF, который означает end of file - конец файла.

Вопрос 176. Как управлять файловой позицией?

Ответ: Мы можем управлять этой файловой позицией с помощью метода seek()

```
file = open("my_file.txt")  
|  
print(file.read(5))  
file.seek(0)  
print(file.read(5))
```

Мы переместили файловую позицию в начало файла. В консоль будет выведено следующее:

Вопрос 177. Как отобразить текущую файловую позицию?

Ответ: С помощью функции tell()

```
file = open("my_file.txt")  
  
print(file.read(5))  
file.seek(0)  
print(file.read(5))  
print(file.tell())
```

Вопрос 178. Как прочитать только строку?

Ответ: С помощью функции `readline()`:

Вопрос 179. Как сделать перебор строк файла?

Ответ:

```
file = open("my_file.txt")

for line in file:
    print(line)
```

Вопрос 180. Как из файла получить список из строк?

Ответ:

```
from pprint import pprint

file = open("my_file.txt")

s = file.readlines()
pprint(s)
```

Вопрос 181. Что надо сделать с файлом, когда мы с ним перестали работать?

Ответ: Его обязательно следует закрыть:

```
file = open("my_file.txt")
|
file.close()
```

Почему? Мы освобождаем память, связанную с этим файлом

Вопрос 182. Как в Python происходит обработка исключений?

Ответ:

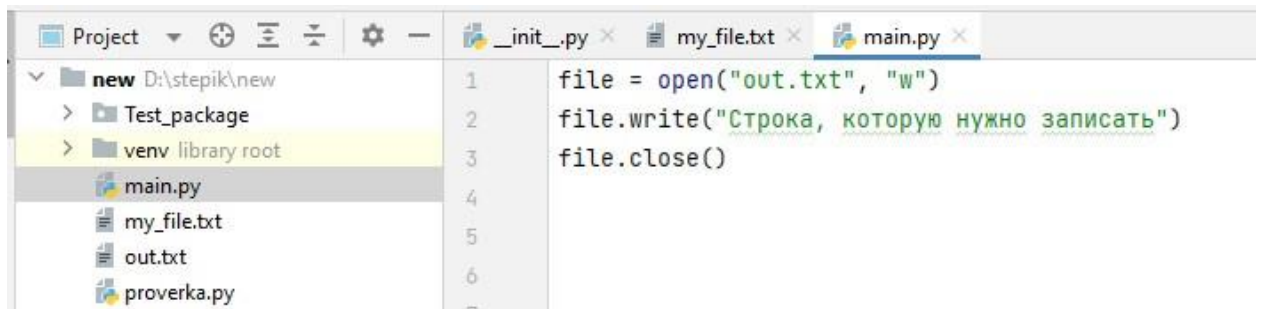
```
try:
    блок операторов
    критического кода
except [исключение]:
    блок операторов
    обработки исключения
finally:
    блок операторов
    всегда исполняемых
    вне зависимости, от
    возникновения исключения
```

Вопрос 183. Что такое ключевое слово with в Python и где используется?

Ответ: with - это ключевое слово, которое является заменой блока try – finally. Используется при работе с файлами

Вопрос 184. Как создать файл?

Ответ: Чтобы создать файл нужно открыть файл на запись:

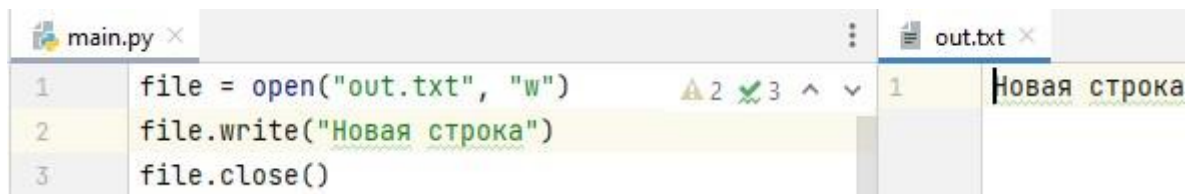


```
1 file = open("out.txt", "w")
2 file.write("Строка, которую нужно записать")
3 file.close()
```

The screenshot shows a project explorer on the left with files: main.py, my_file.txt, out.txt, and proverka.py. The main.py file is open in the editor, showing the code above. The out.txt file is also visible in the project explorer.

Вопрос 185. Как полностью перезаписать файл?

Ответ: Открыть уже существующий файл, поставить метод "w" и в команде write написать новую строку



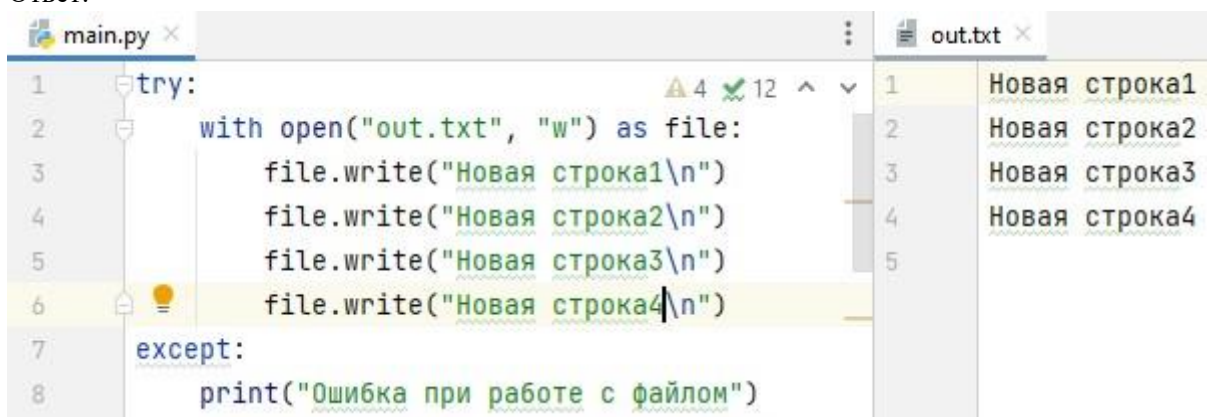
```
1 file = open("out.txt", "w")
2 file.write("Новая строка")
3 file.close()
```

The screenshot shows the main.py file in the editor. The out.txt file is also open in the editor, showing the text "Новая строка".

При этом удаляется все его прежнее содержимое

Вопрос 186. Как записать несколько строчек с новой строки?

Ответ:



```
1 try:
2     with open("out.txt", "w") as file:
3         file.write("Новая строка1\n")
4         file.write("Новая строка2\n")
5         file.write("Новая строка3\n")
6         file.write("Новая строка4\n")
7 except:
8     print("Ошибка при работе с файлом")
```

The screenshot shows the main.py file in the editor. The out.txt file is also open in the editor, showing the text "Новая строка1", "Новая строка2", "Новая строка3", and "Новая строка4".

Если '\n' не будет, то все будет записано в одну строку

Вопрос 187. Как добавить в файл новую информация, не перезаписывая его?

Ответ: Вместо параметра "w" надо записать параметр "a"



The screenshot shows a code editor with two tabs: 'main.py' and 'out.txt'. The 'main.py' tab contains the following code:

```
1 try:
2     with open("out.txt", "a") as file:
3         file.write("New string1\n")
4         file.write("New string2\n")
5 except:
6     print("Ошибка при работе с файлом")
7
```

The 'out.txt' tab shows the output of the script:

```
1 Новая строка1
2 Новая строка2
3 Новая строка3
4 Новая строка4
5 New string1
6 New string2
7
```

Но если такого файла не существует, то он будет заново создан с New string1, New string 2.

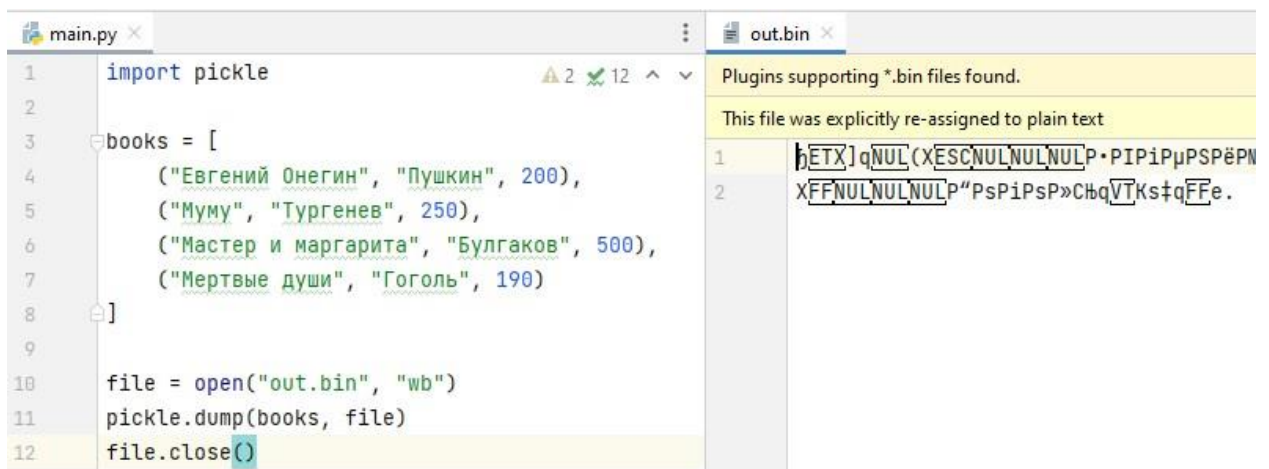
****В записи "w" и перезаписи "a" читать файл нельзя - readline() и readlines() работать не будут!****

Вопрос 188. Что такое бинарный режим работы с файлом?

Ответ: Бинарный режим - это когда, данные из файла считываются без какой-либо обработки. Он используется для сохранения и считывания объектов целиком.

Вопрос 189. Как создать файл в бинарном режиме?

Ответ: Надо импортировать библиотеку pickle, поставить мод на "wb", использовать из библиотеки pickle функцию dump



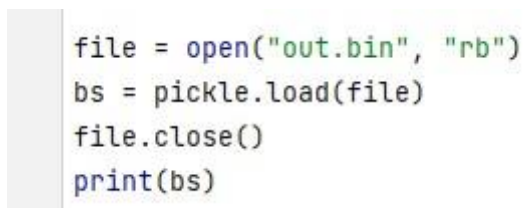
The screenshot shows a code editor with two tabs: 'main.py' and 'out.bin'. The 'main.py' tab contains the following code:

```
1 import pickle
2
3 books = [
4     ("Евгений Онегин", "Пушкин", 200),
5     ("Муму", "Тургенев", 250),
6     ("Мастер и маргарита", "Булгаков", 500),
7     ("Мертвые души", "Гоголь", 190)
8 ]
9
10 file = open("out.bin", "wb")
11 pickle.dump(books, file)
12 file.close()
```

The 'out.bin' tab shows the output of the script, which is a binary file. The output is displayed as a series of hexadecimal characters: `b'ETX]qNUL(XESCNU...PIPIpPSPpP...XFFNU...PsPiPsP»CъqVTks†qFFE.`

Вопрос 190. Как прочитать коллекцию из файла .bin?

Ответ: Надо указать "rb", pickle.load - файл, который мы хотим прочитать, вывести его в отдельную переменную и ее отобразить в консоли



The screenshot shows a code editor with a single tab containing the following code:

```
file = open("out.bin", "rb")
bs = pickle.load(file)
file.close()
print(bs)
```

Вопрос 191. Как в бинарный файл сохранить сразу несколько значений?

Ответ:

```
main.py x
1  import pickle
2
3  book1 = ["Евгений Онегин", "Пушкин", 200]
4  book2 = ["Муму", "Тургенев", 250]
5  book3 = ["Мастер и маргарита", "Булгаков", 500]
6  book4 = ["Мертвые души", "Гоголь", 190]
7
8  try:
9      with open("out.bin", "wb") as file:
10         pickle.dump(book1, file)
11         pickle.dump(book2, file)
12         pickle.dump(book3, file)
13         pickle.dump(book4, file)
14 except:
15     print("Ошибка при работе с файлом")
```

Вопрос 192. Как прочитать данные из бинарного файла?

Ответ:

```
main.py x
1  import pickle
2
3  try:
4      with open("out.bin", "rb") as file:
5         b1 = pickle.load(file)
6         b2 = pickle.load(file)
7         b3 = pickle.load(file)
8         b4 = pickle.load(file)
9  except:
10     print("Ошибка при работе с файлом")
11
12     print(b1, b2, b3, b3, sep="\n")
```

Вопрос 193. Что такое генераторы в Python?

Ответ: Генераторы - это итерируемые объекты, которые не хранят в памяти сразу все значения (в отличие от списков), а генерируют их по мере необходимости, т.е. при переходе к следующему значению. Генераторы используются, чтобы избежать ошибки MemoryError.

Вопрос 194. Как задать объект генератора?

Ответ: Круглые скобки не означают кортеж, генераторов кортежей не существует.

```
a = (x ** 2 for x in range(6))  
print(type(a))
```

Вопрос 195. Как получить конкретное значение из генератора?

Ответ: Генератор сам по себе также является итератором, т.е. его значения можно перебирать с помощью функции next()

```
a = (x ** 2 for x in range(6))  
print(next(a))  
print(next(a))
```

Пока мы не дойдём до самого конца - в конце будет ошибка StopIteration

Вопрос 196. Можно ли использовать цикл for для перебора элементов генератора?

Ответ: Да, можно, т.к. генератор - итерируемый объект

Вопрос 197. Сколько раз можно использовать цикл for для генератора?

Ответ: Только один раз. Второй раз генератор мы перебрать не можем

Вопрос 198. Можно ли генератор превратить в другой тип данных?

Ответ: Да, можно.

- в список с помощью функции list(), но не []
- в множество set{ }

Второй раз уже преобразовать не получится

Вопрос 199. Можно ли использовать функции sum, max, min для генератора?

Ответ: Да, можно, но только один раз

Вопрос 200. Можно ли использовать функцию len() для подсчета количества элементов в генераторе?

Ответ: Нет, не можем. Будет ошибка TypeError.

Вопрос 201. Что такое оператор yield и где он используется?

Ответ: Оператор yield возвращает текущее значение x, замораживает состояние функции, все переменные внутри функции замораживают свое состояние до следующего вызова функции next().

```
main.py x
1 def get_list():
2     for x in [1,2,3,4]:
3         yield x
4
5 a = get_list()
6 print(a)
```

Перебирать значения можно через функцию next()

Вопрос 202. Что такое функция map и как она работает?

Ответ: Функция map - это генератор, который нужен для выполнения какой-нибудь функции над элементами итерируемого объекта.

Например,

```
`b = map(int,['1', '2', '3'])`
```

это тоже самое, что и

```
`b = (int(i) for i in ['1', '2', '3'])`
```

Часто в функции map используются lambda-функции:

```
`b = list(map(lambda s: s[:-1], cities))`
```

Задом наперед названия городов

Вопрос 203. Что такое функция filter и как она работает?

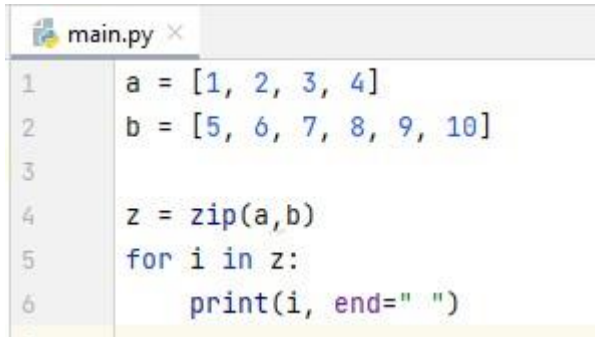
Ответ: Функция filter служит для отбора элементов из указанного итерируемого объекта в том случае, если func равно True.

```
main.py x
1 s = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
3 b = filter(lambda x: x % 2 == 0, s)
4 for i in b:
5     print(i, end=" ")
6
```

```
main.py x
1 s = ['cat', 'dog', 'horse', 'catty', 'catous']
2
3 my_cat = filter(lambda x: True if 'cat' in x else False, s)
4 for i in my_cat:
5     print(i, end=" ")
6
```

Вопрос 204. Что такое функция zip и как она работает?

Ответ: Функция zip - это такая функция, которая для указанных итерируемых объектов выполняет перебор соответствующих элементов и продолжает работать до тех пор, пока не дойдет до конца самой короткой коллекции.



```
main.py x
1 a = [1, 2, 3, 4]
2 b = [5, 6, 7, 8, 9, 10]
3
4 z = zip(a,b)
5 for i in z:
6     print(i, end=" ")
7
```

Результат:

`(1, 5) (2, 6) (3, 7) (4, 8)`

Вопрос 205. Что такое функция isinstance и как она работает?

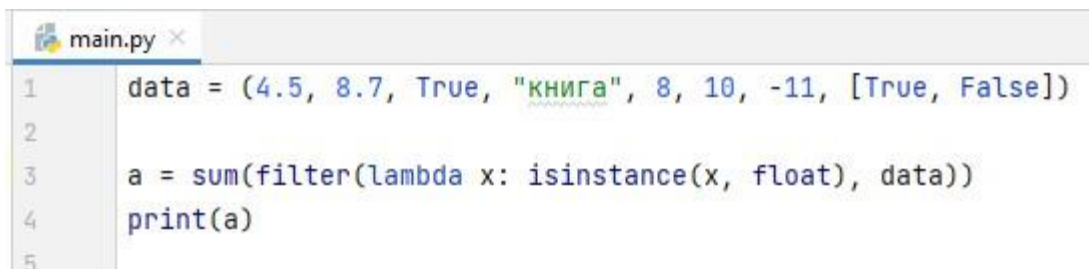
Ответ: Производит проверку объекта определенному классу.

Возвращает True, если объект соответствует определенному классу (типу данных)

False - если не соответствует

****Не следует применять isinstance для типа bool, т.к. тип int наследуется от типа bool и isinstance покажет, что целое число относится к булевому типу, что неверно****

****Поэтому для int и bool делается строгая проверка type(a) is bool****



```
main.py x
1 data = (4.5, 8.7, True, "книга", 8, 10, -11, [True, False])
2
3 a = sum(filter(lambda x: isinstance(x, float), data))
4 print(a)
5
```

Результат:

`13.2`

Вопрос 206. Что такое функция all() и как она работает?

Ответ: Функция all проверяет, что в итерируемом объекте все элементы имели булево значение True. Если хотя бы один объект имеет False то all вернет False.

```
main.py x
1 lst = [True, True, True, 5, 5.6, [1, 2, 3], "+", (5, 6, 7,)]
2 print(all(lst))
3
```

Здесь нет False и пустых значений, и нулей

Результат:

`True`

Вопрос 207. Что такое функция any() и как она работает?

Ответ: Функция any проверяет, что в итерируемом объекте все элементы имели булево значение False. Если хотя бы один объект имеет True то any вернет True.

```
main.py x
1 lst = [0, 0.0, {}, [], "", ()]
2 print(any(lst))
3
```

Все значения списка имеют False. Поэтому any вернет False

Результат:

`False`

Вопрос 208. Что такое конструкция match-case и где используется?

Ответ: Конструкция match-case позволяет гибко анализировать переменные на соответствие шаблонов и для найденного совпадения выполнять заданные операции. Все тоже самое можно сделать и с помощью if-elif-else, но match-case имеет ряд преимуществ при работе с коллекциями , а также уменьшает количество кода.

Как только один из блоков case отработал, выполняются операторы этого case, происходит выход из конструкции match-case

```

cmd = "top"

match cmd:
    case "top":
        print("вверх")
    case "left":
        print("влево")
    case "right":
        print("вправо")
    case _: # wildcard
        print("другое")

print("проверки завершены")

```

В конструкции match-case обязательно должны быть операторы после слова case и двоеточия. Блок case _ называется wildcard, он используется, когда ни один выше case не сработал. Его можно прописывать, а можно не прописывать.

Вопрос 209. Что такое мануальное тестирование программы Python?

Ответ: Мануальное тестирование - это ручная проверка, когда проигрываются все возможные сценарии поведения программы.

Здесь работают по пользовательским сценариям, выполняют определенные действия и смотрят на результат.

В случае, когда результат отличается от ожидаемого - тестировщик пишет отчет об ошибке, bug-report. По итогам полной проверки тестировщик сдает test-report.

Получив test-report разработчик исправляет ошибки и передаёт проект на повторное тестирование.

Вопрос 210. Что такое assert и когда используется?

Ответ: в Python есть встроенная инструкция assert. (assert - утверждение)

Логика работы assert:

- Разработчик передаёт в него некоторое утверждение
- Если оно истинно, assert ничего не возвращает. Тест пройден
- Если утверждение оказалось ложным выбрасывает исключение, с сообщением об ошибке, а само выполнение кода прерывается.

`assert <проверяемое утверждение>, <Сообщение об ошибке>`

```

assert test > fun.py
1  assert 4 > 7, 'Неверно'

```

Вопрос 211. Какие библиотеки в Python существуют для написания автотестов кода?

Ответ: docktest, pytest, unittest

Вопрос 212. Что такое TDD?

Ответ: TDD - Test-drive Development (разработка через тестирование)

Принцип:

1. Написать тесты
 2. Написать код, соответствующий этим тестам
-

Вопрос 213. Что такое паттерн AAA при написании автотестов?

Ответ: Arrange (настройка) — в этом блоке кода мы подготавливаем данные для теста. _Обычно это создание экземпляра класса тестируемого юнита._

Act — выполнение или вызов тестируемого сценария.

Assert — проверка, что тестируемый вызов ведёт себя ожидаемо.

Такой подход называется паттерн AAA. Он улучшает структуру теста и его читабельность.

Вопрос 214. Что такое фикстуры при тестировании?

Ответ: test fixtures - это фиксированные объекты и данные для выполнения тестов. Перед началом теста в коде создаются объекты и данные, на которых будет проведено тестирование. Фикстурами могут быть:

- содержимое баз данных
- набор переменных среды
- набор файлов с необходимым содержанием

Фикстуры передаются в тест при запуске тестирования и никак не влияют на данные проекта.

Вопрос 215. Для чего используются стандартные библиотеки time и datetime. В чем их отличие?

Ответ: Библиотека time используется для обращения к системным функциям, относящимся ко времени. Библиотека datetime позволяет манипулировать датой и временем. Например, выполнять арифметические операции, сравнивать даты, выводить дату и время в различных форматах.

Вопрос 216. Для чего используется стандартная библиотека random?

Ответ: Этот модуль реализует генераторы псевдослучайных чисел для различных дистрибутивов.

Вопрос 217. Что такое функция reduce и для чего она используется?

reduce - это функция встроенного модуля functools в Python, которая используется для последовательного применения функции к элементам итерируемого объекта, с целью свести их к одному значению.

Основная сигнатура функции reduce выглядит следующим образом:

```
functools.reduce(function, iterable[, initializer])
```

function: это функция, которая будет применяться к элементам последовательности.

iterable: итерируемый объект, значения которого последовательно обрабатываются функцией.

initializer (опциональный): начальное значение, которое используется перед первым элементом последовательности.

Пример использования reduce для нахождения суммы элементов списка:

```
from functools import reduce

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

# Функция, складывающая два числа
sum_func = lambda x, y: x + y

# Используем reduce для нахождения суммы элементов списка
result = reduce(sum_func, numbers)

print(result) # Вывод: 15
```

В данном примере reduce применяет функцию sum_func последовательно ко всем элементам списка, сводя их к единственному значению - сумме элементов.

Обратите внимание, что начальное значение (initializer) не обязательно. Если оно не указано, то первые два элемента итерируемого объекта будут использованы в качестве начальных значений.