



Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Колледж космического машиностроения и технологий

ОТЧЕТ

По учебной практике УП.01.01 Разработка программных модулей
программного обеспечения для компьютерных систем
специальность 09.02.03 Программирование в компьютерных системах

Выполнили студенты:

Брусник Вадим Виктрович

_____ (подпись)

Мырза Николай Юрьевич

_____ (подпись)

Гусятинер Леонид Борисович

_____ (подпись)

_____ (оценка)

Королев, 2020

Содержание:

| | |
|---|----------|
| Раздел 1. Техника решения задач с использованием структурного и объектно-ориентированного программирования | 2 |
| 1.1 Установка интерпретатора Python 3 | 2 |
| 1.2. Техника работы в командной строке и среде IDLE программами..... | 5 |
| 1.3 Техника работы с линейными и разветвляющимися программами | 7 |
| 1.4 Техника работы с циклическими программами, цикл while | 11 |
| 1.5 Техника работы с числами | 13 |
| 1.6 Техника работы со строками | 15 |
| 1.7 Техника работы со списками | 16 |
| 1.8 Техника работы с циклом for и генераторами списков..... | 21 |
| 1.9 Техника работы с функциями..... | 24 |
| 1.10 Техника работы со словарями..... | 27 |
| 1.11 Техника работы с множествами..... | 31 |
| 1.12 Техника работы с кортежами | 32 |
| 1.13 Техника работы с файлами..... | 34 |
| 1.14 Техника работы с модулями..... | 35 |
| 1.15 Техника работы с классами | 60 |

Раздел 1. Техника решения задач с использованием структурного и объектно-ориентированного программирования

1.1 Установка интерпретатора Python 3

Для установки интерпретатора Python на компьютер, первое, что нужно сделать – это скачать дистрибутив. Загрузить его можно с официального сайта, перейдя по ссылке <https://www.python.org/downloads/>

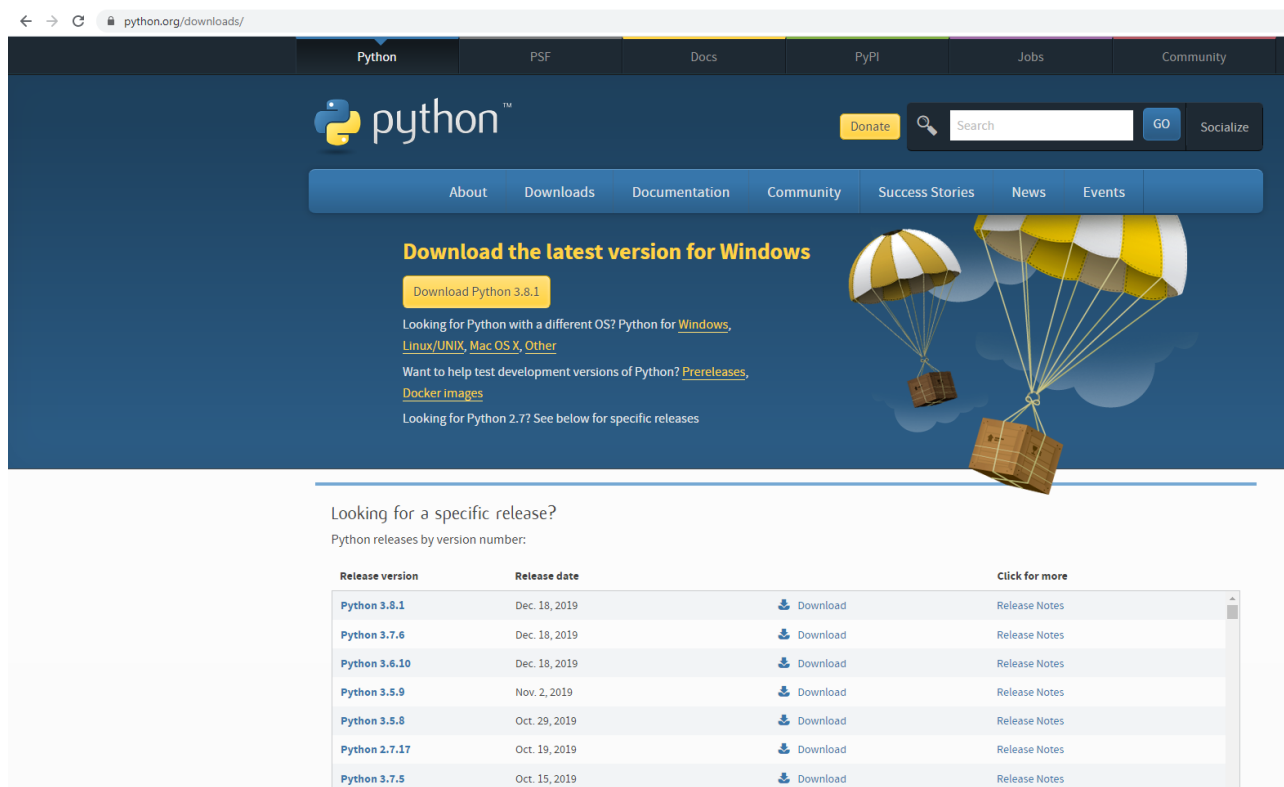


Рис.1.Официальный сайт Python

Порядок установки на Windows:

1. Запустить скачанный установочный файл.
2. Выбрать способ установки.

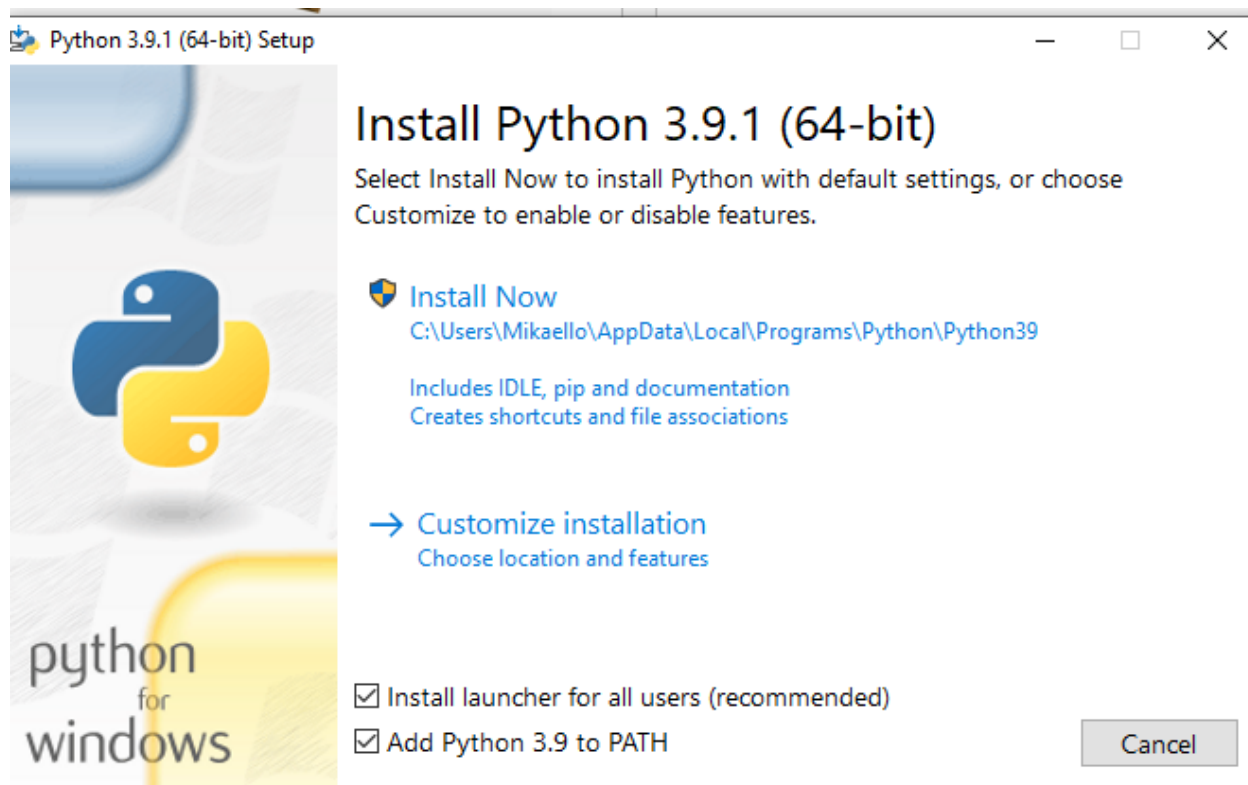


Рис.2. Установщик Python

3. Отметить необходимые опции установки (доступно при выборе Customize installation)



Рис.3. Опции установки

На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Выбираю:

- Documentation – установка документаций.
- pip – установка пакетного менеджера pip.
- tcl/tk and IDLE – установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).

4. Выбираем место установки (доступно при выборе Customize installation)

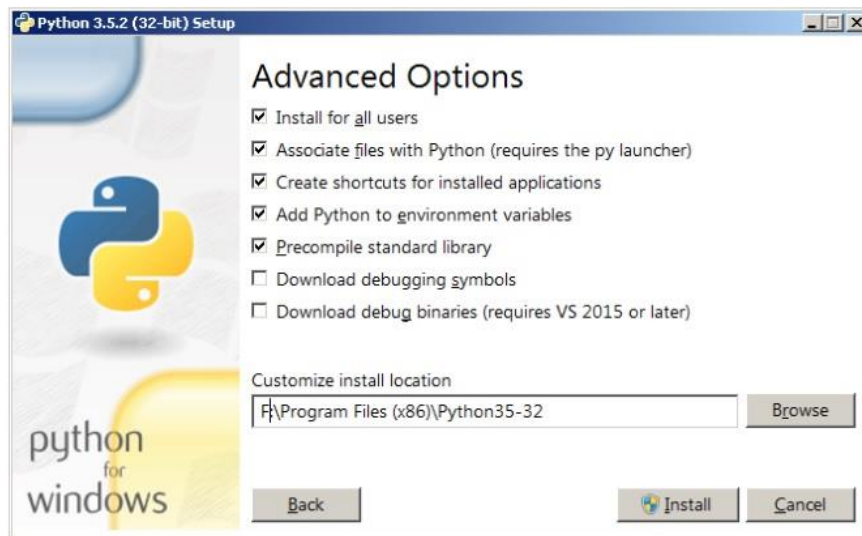


Рис.4.Продвинутые опции установки

5. После успешной установки:



Рис.5.Результат успешной установки

1.2. Техника работы в командной строке и среде IDLE программами

Выполняя (запуская) команду “python” в вашем терминале, вы получаете интерактивную оболочку Python.

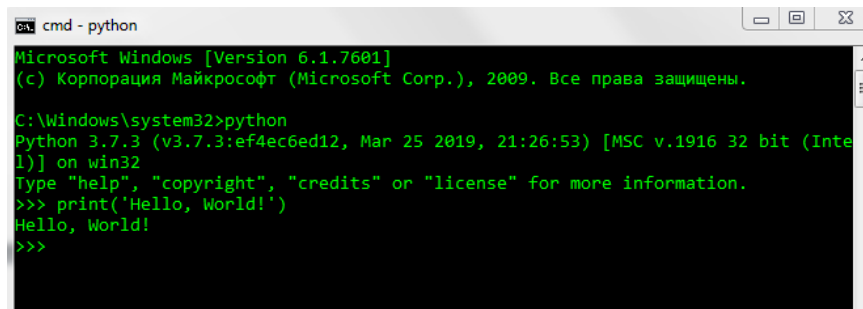


Рис.1.Интерактивная оболочка Python

Существует несколько способов закрыть оболочку Python:

```
>>> exit()
```

или же

```
>>> quit()
```

Кроме того, **CTRL + D** закроет оболочку и вернет вас в командную строку терминала.

IDLE - простой редактор для Python, который поставляется вместе с Python.

Откройте IDLE в вашей системе выбора.

В оболочке есть подсказка из трех прямоугольных скобок:

```
>>>
```

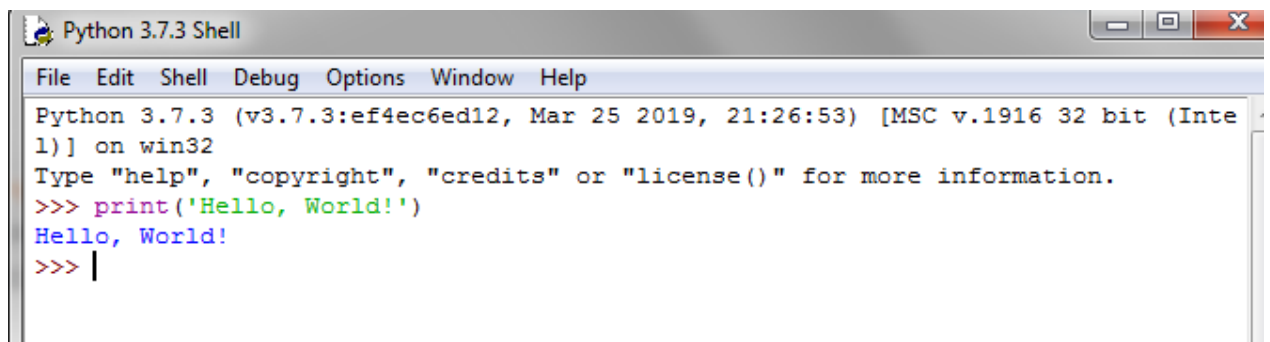
Теперь напишите в подсказке следующий код:

```
>>> print("Hello, World")
```

Нажмите **Enter**.

```
>>> print("Hello, World")
```

```
Hello, World
```



The image shows a screenshot of a Python 3.7.3 Shell window. The window has a title bar that says "Python 3.7.3 Shell" and standard Windows window controls (minimize, maximize, close). Below the title bar is a menu bar with the following options: File, Edit, Shell, Debug, Options, Window, and Help. The main area of the window contains the following text:

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello, World!')
Hello, World!
>>> |
```

Рис.2. Первая программа

1.3 Техника работы с линейными и разветвляющимися программами

Листинг 1. НОК

```
'''
```

С использованием результата задания 2 разработать программу для нахождения наименьшего общего кратного

```
'''
```

```
a = int(input())
b = int(input())
i = min(a, b)

while True:
    if i%a == 0 and i%b == 0:
        break
    i += 1

print(i)
```

Листинг 2.input и print

```
'''
```

```
- input
Функция input() в Python, ввод данных с клавиатура.
https://docs-python.ru/tutorial/vstroennye-funktsii-interpretatora-python/funktsija-input/
'''
strenk = input()
print(strenk)
```

Листинг 3.Меню

```
'''
```

Задание 2. Разработать программу с меню для демонстрации работы с типами данных:

список(list), словарь(dict), множество(set)

Меню -> выбор типа данных -> выбор метода -> краткая справка

```
'''
```

```
options = int(input("Choose from Menu:\n"
                    "list: 1.\n"
                    "dict: 2.\n"
                    "set: 3.\n"))
```

```
listMethods = ['list.append(x)      Добавляет элемент в конец
списка\n',
```



```

'list.extend(L)      Расширяет список list, добавляя в конец все
элементы списка L\n',
'list.insert(i, x)   Вставляет на i-ый элемент значение x\n',
'list.remove(x)      Удаляет первый элемент в списке, имеющий
значение x. ValueError, если такого элемента не существует\n'
'list.pop([i])       Удаляет i-ый элемент и возвращает его. Если индекс
не указан, удаляется последний элемент\n'
'list.index(x, [start [, end]]) Возвращает положение первого
элемента со значением x (при этом поиск ведется от start до
end)\n'
'list.count(x)       Возвращает количество элементов со значением x\n'
'list.sort([key=функция]) Сортирует список на основе
функции\n'
'list.reverse()      Разворачивает список\n'
'list.copy()         Поверхностная копия списка\n'
'list.clear()        Очищает список\n']

```

```
dictMethods = ['dict.clear() - очищает словарь.\n',
```

```
'dict.copy() - возвращает копию словаря.\n',
```

```
'classmethod dict.fromkeys(seq[, value]) - создает словарь с
ключами из seq и значением value (по умолчанию None).\n',
```

```
'dict.get(key[, default]) - возвращает значение ключа, но если
его нет, не бросает исключение, а возвращает default (по
умолчанию None).\n',
```

```
'dict.items() - возвращает пары (ключ, значение).\n',
```

```
'dict.keys() - возвращает ключи в словаре.\n',
```

```
'dict.pop(key[, default]) - удаляет ключ и возвращает значение.
Если ключа нет, возвращает default (по умолчанию бросает
исключение).\n',
```

```
'dict.popitem() - удаляет и возвращает пару (ключ, значение).
Если словарь пуст, бросает исключение KeyError. Помните, что
словари неупорядочены.\n',
```

```
'dict.setdefault(key[, default]) - возвращает значение ключа, но
если его нет, не бросает исключение, а создает ключ с значением
default (по умолчанию None).\n',
```

```

'dict.update([other]) - обновляет словарь, добавляя пары (ключ,
значение) из other. Существующие ключи перезаписываются.
Возвращает None (не новый словарь!).\n',

'dict.values() - возвращает значения в словаре.\n']

setMethods = ['len(s) - число элементов в множестве (размер
множества).\n',
'x in s - принадлежит ли x множеству s.\n',
'set.isdisjoint(other) - истина, если set и other не имеют общих
элементов.\n',
'set == other - все элементы set принадлежат other, все элементы
other принадлежат set.\n',
'set.issubset(other) или set <= other - все элементы set
принадлежат other.\n',
'set.issuperset(other) или set >= other - аналогично.\n',
'set.union(other, ...) или set | other | ... - объединение
нескольких множеств.\n',
'set.intersection(other, ...) или set & other & ... -
пересечение.\n',
'set.difference(other, ...) или set - other - ... - множество из
всех элементов set, не принадлежащие ни одному из other.\n',
'set.symmetric_difference(other); set ^ other - множество из
элементов, встречающихся в одном множестве, но не встречающиеся
в обоих.\n',
'set.copy() - копия множества.\n',
'И операции, непосредственно изменяющие множество:\n',

'set.update(other, ...); set |= other | ... - объединение.\n',
'set.intersection_update(other, ...); set &= other & ... -
пересечение.\n',
'set.difference_update(other, ...); set -= other | ... -
вычитание.\n',
'set.symmetric_difference_update(other); set ^= other -
множество из элементов, встречающихся в одном множестве, но не
встречающиеся в обоих.\n',
'set.add(elem) - добавляет элемент в множество.\n',
'set.remove(elem) - удаляет элемент из множества. KeyError, если
такого элемента не существует.\n',
'set.discard(elem) - удаляет элемент, если он находится в
множестве.\n',
'set.pop() - удаляет первый элемент из множества. Так как
множества не упорядочены, нельзя точно сказать, какой элемент
будет первым.\n',
'set.clear() - очистка множества.\n']

```

```
menuList = [listMethods, dictMethods, setMethods]

print(*menuList[options - 1])
```

Листинг 4. форматная строка и метод формат

```
'''
- форматная строка и метод формат
'''
print('Hello, {0}, {1}!'.format(input(), input()))
```

Листинг 5. программа для печати даты прописью

```
'''
- Задание 1. Разработать программу для печати даты прописью
Пример ввода: 15.12.1983
Пример вывода: Пятнадцатое декабря одна тысяча девятсот
восемьдесят третьего года
'''

def get_date(date):
    days = ['первое', 'второе', 'третье', 'четвёртое',
            'пятое', 'шестое', 'седьмое', 'восьмое',
            'девятое', 'десятое', 'одиннадцатое', 'двенадцатое',
            'тринадцатое', 'четырнадцатое', 'пятнадцатое',
            'шестнадцатое',
            'семнадцатое', 'восемнадцатое', 'девятнадцатое',
            'двадцатое',
            'двадцать первое', 'двадцать второе', 'двадцать третье',
            'двадцать четвертое', 'двадцать пятое', 'двадцать
шестое',
            'двадцать седьмое', 'двадцать восьмое', 'двадцать
девятое',
            'тридцатое', 'тридцать первое']

    months = ['января', 'февраля', 'марта', 'апреля', 'мая',
              'июня',
              'июля', 'августа', 'сентября', 'октября', 'ноября',
              'декабря']

    date = date.split('.')
    return (days[int(date[0]) - 1] + ' ' + months[int(date[1]) -
1] + ' ' + date[2] + ' года')

date = input()
print(get_date(date))
```

1.4 Техника работы с циклическими программами, цикл while

Листинг 5.Sum

```
'''
```

Напишите программу, которая считывает со стандартного ввода целые числа, по одному числу в строке, и после первого введенного нуля выводит сумму полученных на вход чисел.

```
'''
```

```
sum = 0
a=int(input())
while a !=0:
    sum +=a
    a=int(input())
print(sum)
```

Листинг 6.Последовательность

```
'''
```

Напишите программу, которая выводит часть последовательности 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5 ...

(число повторяется столько раз, чему равно).

На вход программе передаётся неотрицательное целое число n – столько элементов

последовательности должна отобразить программа.

На выходе ожидается последовательность чисел, записанных через пробел в одну строку.

Например, если n = 7, то программа должна вывести 1 2 2 3 3 3 4.

```
'''
```

```
n = int(input())
a = []
i = 1
while len(a) < n:
    a +=[i] * i
    i +=1
print(*a[:n])
```

Листинг 7.While

```
'''
```

<https://stepik.org/lesson/3364/step/11?unit=947>

Напишите программу, которая считывает со стандартного ввода целые числа, по одному числу

в строке, и после первого введенного нуля выводит сумму полученных на вход чисел.

```
'''
```

```
b = int(input())
```

```

s = b
while b != 0:
    c = int(input())
    s += c
    b = c
print(s)

```

Листинг 8.LCM

```

'''
С использованием результата задания 2 разработать программу для
нахождения наименьшего
общего кратного
'''
a = int(input())
b = int(input())
i = min(a, b)

while True:
    if i%a==0 and i%b==0:
        break
    i += 1

print(i)

```

1.5 Техника работы с числами

Листинг 9. Fractions and decimal

```
'''
Задание 1. Составить и выполнить по 3 примера использования
модулей для работы
с дробными числами (fractions), для точных вычислений (decimal).
'''

from decimal import Decimal
from fractions import Fraction

number = Decimal("0.444")
print(number)
number = number.quantize(Decimal("1.00"))
print(number)

number = Decimal("0.555678")
print(number.quantize(Decimal("1.00")))

number = Decimal("0.999")
print(number.quantize(Decimal("1.00")))

print(Fraction(1, 3))

print(Fraction(3.1415))

a = Fraction(1, 7)
b = Fraction(1, 3)
print(a + b)
```

Листинг 10. Больше Соседей

```
'''
https://pythontutor.ru/lessons/lists/problems/more\_than\_neighbours/
Задача «Больше своих соседей»
Дан список чисел. Определите, сколько в этом списке элементов,
которые больше двух
своих соседей, и выведите количество таких элементов. Крайние
элементы списка никогда
не учитываются, поскольку у них недостаточно соседей.
'''

arr = []
```

```
size = int(input())
count = 0
for i in range(size):
    arr.append(int(input()))
for i in range(size):
    if i > 0 and i < size - 1:
        if arr[i - 1] < arr[i] and arr[i] > arr[i + 1]:
            count += 1

print(count)
```

1.6 Техника работы со строками

Листинг 11. Убрать точки

```
'''
?Уберите точки из введенного IP-адреса. Выведите сначала четыре
числа через пробел,
а затем сумму получившихся чисел.
Sample Input: 192.168.0.1
Sample Output:
192 168 0 1
361
'''

string = input().replace('.', ' ')
print(string)

string = string.split()

print(int(string[0]) + int(string[1]) + int(string[2]) +
int(string[3]))
```

Листинг 12. replace

```
'''
?Известно, что для логина часто не разрешается использовать
строки содержащие пробелы.
Но пользователю нашего сервиса особенно понравилась какая-то
строка.
Замените пробелы в строке на символы нижнего подчеркивания,
чтобы строка
могла стодиться для логина. Если строка состоит из одного слова,
менять ничего не нужно.
Sample Input: python sila
Sample Output: python_sila
'''

print(input().replace(' ', '_'))
```


1.7 Техника работы со списками

Листинг 12. Array114

```
'''
Array114. Дан массив A размера N. Упорядочить
его по возрастанию методом сортировки простыми вставками:
сравнить элементы A0 и A1 и, при необходимости меняя их
местами, добиться того, чтобы они оказались упорядоченными
по возрастанию; затем обратиться к элементу A2 и
переместить его в левую (уже упорядоченную) часть массива,
сохранив ее упорядоченность; повторить этот процесс для
остальных элементов, выводя содержимое массива после
обработки каждого элемента (от 1-го до N-1 го).
'''
'''
10
10
9
8
7
6
5
4
3
2
1
'''
arr = []
size = int(input())

for i in range(size):
    arr.append(int(input()))

for i in range(size - 1):
    for j in range(size - 1 - i):
        MIN, MAX = min(arr[j], arr[j + 1]), max(arr[j], arr[j +
1])
        arr[j], arr[j + 1] = MIN, MAX

print(arr)
```

Листинг 13. Array112

```
'''
```

Array112. Дан массив A размера N.
 Упорядочить его по возрастанию методом сортировки
 простым обменом («пузырьковой» сортировкой):
 просматривать массив, сравнивая его соседние элементы
 (A0 и A1, A1 и A2 и т. д.) и меняя их местами,
 если левый элемент пары больше правого; повторить описанные
 действия N - 1 раз. Для контроля за выполняемыми действиями
 выводить содержимое массива после каждого просмотра.
 Учесть, что при каждом просмотре количество анализируемых
 пар можно уменьшить на 1.

```
'''
N = int(input())
array = []
for x in range(N):
    array.append(int(input()))
print(*array)

for i in range(N - 1):
    for j in range(N - i - 1):
        if array[j] > array[j + 1]:
            array[j], array[j + 1] = array[j + 1], array[j]
            print(*array)
print("\n Итог:")
print(*array)
'''
```

Листинг 14. Matrix88

'''
 Matrix88. Дана квадратная матрица порядка M. Обнулить элементы
 матрицы,
 лежащие ниже главной диагонали. Условный оператор не
 использовать.

```
4
4
1
1
1
1
4

4
4
4
2
2
2
```

```

2
3
3
3
3
'''

# в первой строке ввода идёт количество строк массива
m, n = int(input()), int(input())
a = []
for i in range(m):
    b = []
    for j in range(n):
        b.append(int(input()))
    a.append(b)

for i in range(m):
    for j in range(n - i, n):
        a[i][j] = 0

for i in range(m):
    print(*a[i])

```

Листинг 15.Matrix3

```

'''
Задание 4. Matrix3. Даны целые положительные числа M, N и набор
из M чисел. Сформировать
матрицу размера M x N, у которой в каждом столбце содержатся все
числа из исходного
набора (в том же порядке).
'''

# в первой строке ввода идёт количество строк массива
m = int(input())
b = []
for i in range(m):
    b.append(int(input()))
print("NEXT\n")
n = int(input())
a = []
for i in range(m):
    a.append([0] * n)

for i in range(m):
    for j in range(n):
        a[i][j] = b[i]

```

```
print(*a)
```

Листинг 16.

```
'''
Задание 1. (Л.Б.) Для проведения конкурса проектов в ККМТ
формируются группы
из 4х участников: coder, writer, tester, designer,
программирующих
на одном и том же языке.
Каждый студент может программировать только на одном языке
и занимать только одну позицию.
Дан текстовый файл, содержащий перечень студентов с указанием
языка и позиции
(каждый студент с новой строки)
Требуется
1. Получить список студентов с указанием языка и позиции
2. Сформировать список всевозможных команд
3. Вывести список команд с указанием состава и названия команды:
Команда 1
coder: ...
designer: ...
tester: ...
writer: ...

Команда 2
...
Пункты 1 и 2 выполнить с использованием генераторов списка
'''

listOfPosition = ['coder:', 'designer:', 'tester:', 'writer:']

listOf_FirstTeam = []

count = 0

s = str

i = 0

f = open('file.txt', 'r')
with open('file.txt', 'r') as file:
    for line in file:
        s = line
        s = s.split()
        position = s[0]
```

```
        for i in range(4):
            if position == listOfPosition[i]:
                listOf_FirstTeam.append(s)

print(*listOf_FirstTeam)

print(count)
```

1.8 Техника работы с циклом for и генераторами списков

Листинг 17.Array55

```
'''
```

Задание 1. Array55. Дан целочисленный массив A размера N (≤ 15). Переписать в новый целочисленный массив B все элементы с нечетными порядковыми номерами (1, 3, ...) и вывести размер полученного массива B и его содержимое. Условный оператор не использовать.

```
'''
```

```
n = int(input())
a = [int(input()) for _ in range(n)]
b = [x for x in a if x % 2 == 1]
print(len(b))
print(*b)
```

Листинг 18.Array57

```
'''
```

Задание 2. Array57. Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B того же размера вначале все элементы исходного массива с четными номерами, а затем – с нечетными:
A[0], A[2], A[4], A[6], ..., A[1], A[3], A[5],
Условный оператор не использовать.

```
'''
```

```
n = int(input())
a = [int(input()) for _ in range(n)]
b = []
for i in range(0, n, 2):
    b.append(a[i])
for i in range(1, n, 2):
    b.append(a[i])
print(*b)
```

Листинг 19.Array58

```
'''
```

Задание 3. Array58. Дан массив A размера N. Сформировать новый массив B того же размера по следующему правилу: элемент B[K] равен сумме элементов массива A с номерами от 0 до K.

```
'''
```

```

n = int(input())
a = [int(input()) for _ in range(n)]
b=[]
sums = 0
for i in range(0,len(a)):
    sums += a[i]
    b.append(sums)
print(*b)

```

```
'''
```

Техника работы с циклом for и генераторами списков

Листинг 20. Распределение по файлам.

```
'''
```

Задание 1. (Л.Б.) Для проведения конкурса проектов в ККМТ формируются группы из 4х участников: coder, writer, tester, designer, программирующих на одном и том же языке.

Каждый студент может программировать только на одном языке и занимать только одну позицию.

Дан текстовый файл, содержащий перечень студентов с указанием языка и позиции(каждый студент с новой строки)

Требуется:

1. Получить список студентов с указанием языка и позиции
2. Сформировать список всевозможных команд
3. Вывести список команд с указанием состава и названия команды:

Команда 1

coder: ...

designer: ...

tester: ...

writer: ...

Команда 2

```
...
```

Пункты 1 и 2 выполнить с использованием генераторов списка

```
'''
```

```

file = open("file.txt", "r")
def prof(mtrx,mas):
    if (len(mtrx) == 0):
        return True
    count = 0
    for i in range(len(mtrx)):
        if(mtrx[i][1] != mas[1]):
            count += 1
        if(mtrx[i][2] == mas[2]):
            count += 1

```

```

        if(count == len(mtrx)*2):
            return True
        return False

list = file.readlines()

out = [i.strip().split() for i in list]
for i in range(len(out)):
    print(f'{out[i][1]}: {out[i][0]} | {out[i][2]}')
print()
print(out)
print('Всего участников - ', len(out), '\n')
teams = [[]]

k = 0
count = 0
i = 0
while(i != len(out)):
    if(prof(teams[k],out[i])):
        if(len(teams[k]) < 4):
            teams[k].append(out[i])
            out.pop(i)
            i -= 1
            if(len(teams[k]) == 4):
                teams.append([])
                k += 1
                i = 0
                continue

        i += 1
    if(i == len(out)):
        teams.append([])
        k += 1
        i = 0
teams.pop(-1)

l = 0
for i in range(len(teams)):
    if(len(teams[i]) < 4):
        print("Неполная команда")
    else:
        l += 1
        print("Команда", l)
        for j in range(len(teams[i])):
            print(f'{teams[i][j][1]}: {teams[i][j][0]} | {teams[i][j][2]}')
        print()

```


1.9 Техника работы с функциями

Листинг 21. Lambda, map

```
'''
```

Использовать lambda, map.

<https://stepik.org/lesson/239422/step/2?unit=211833>

Быстрая инициализация. Программа получает на вход три числа через пробел — начало и конец

диапазона, а также степень, в которую нужно возвести каждое число из диапазона. Выведите

числа получившегося списка через пробел.

Sample Input:

5 12 3

Sample Output:

125 216 343 512 729 1000 1331 1728

3

5

12

```
'''
```

```
degree = int(input())
```

```
result = list(map(lambda x: x ** degree, range(int(input()),  
int(input()) + 1)))
```

```
print(*result)
```

Листинг 22.SortInc

```
'''
```

Func33. Описать функцию SortInc3(X), меняющую содержимое списка X из трех вещественных элементов таким образом, чтобы их значения оказались упорядоченными по возрастанию (функция возвращает None). С помощью этой функции упорядочить по возрастанию два данных списка X и Y.

```
'''
```

```
import math
```

```
def sortInc3(X):
```

```
#     X - list for next sort.
```

```
    for i in range(len(X) - 1):
```

```
        for j in range(len(X) - i - 1):
```

```
            if X[j] > X[j + 1]:
```

```
                X[j], X[j + 1] = X[j + 1], X[j]
```

```
    return None
```

```
list = []
```

```
for i in range(3):
```

```
list.append(float(input()))

list2 = []
for i in range(3):
list2.append(float(input()))

sortInc3(list)
sortInc3(list2)

print(*list)
print(*list2)
```

Листинг 23.Lambda, Filter.

```
'''
Использовать lambda, filter.
Array55. Дан целочисленный массив А размера N (<= 15).
Переписать в новый целочисленный
массив В все элементы с нечетными порядковыми номерами (1, 3,
...) и вывести размер
полученного массива В и его содержимое. Условный оператор не
использовать.
1
2
3
4
5
.
'''

l = [int(x) for x in iter(input, '.')]
indexes = filter(lambda i: i % 2 != 0, range(len(l)))
result = [l[i] for i in indexes]
print(*result)
```

Листинг 24.Map, Lambda.

```
'''
https://stepik.org/lesson/201702/step/13?unit=175778
Использовать map, lambda
Квадраты в обратном порядке. Числа вводятся до точки. Через
пробел выведите эти числа в
обратном порядке, возводя их в квадрат.
Sample Input:
5
16
20
```

```
1
9
.
```

Sample Output:

```
81 1 400 256 25
```

```
'''
```

```
import math
```

```
l = [int(x) for x in iter(input, '.')]
result = list(map(lambda x: print(x ** 2, end=' '), l[::-1]))
```

Листинг 25.IsSquare.

```
'''
```

Func10. Описать функцию IsSquare(K) логического типа, возвращающую True, если целый параметр K (> 0) является квадратом некоторого целого числа, и False в противном случае. С ее помощью найти количество квадратов в наборе из 10 целых положительных чисел.

```
'''
```

```
def isSquare(K):
    # K - int Number .
    i = 0
    while K >= i ** 2:
        i += 1
        if (i ** 2 == K):
            return True
    return False
```

```
num = int(input())
print(isSquare(num))
```

1.10 Техника работы со словарями

Листинг 26.dict.

```
'''
https://pythontutor.ru/lessons/dicts/problems/occurency\_index/
Задача «Номер появления слова»
Условие. В единственной строке записан текст. Для каждого слова
из данного текста
подсчитайте, сколько раз оно встречалось в этом тексте ранее.
Словом считается последовательность непробельных символов идущих
подряд, слова разделены
одним или большим числом пробелов или символами конца строки.

New year New year yeeah!
'''
s = input()
s = s.split(' ')
print(*s)
d = {

}
for i in range(len(s)):
    rem = s[i]
    d[i] = s[i]
print(d)
list = []
for i in range(len(d)):
    count = 1
    for j in range(i + 1, len(d)):
        if d[i] == d[j] and d[i] != 'None':
            count += 1
            d[j] = 'None'
    if count > 0 and d[i] != 'None':
        print(s[i], ': ', count)
    count = 0
```

Листинг 27. Телефонная книга.

```
'''
https://stepik.org/lesson/243394/step/4?unit=215740
Телефонная книга. Этап 1. Коля устал запоминать телефонные
номера и заказал у Вас
программу, которая заменила бы ему телефонную книгу. Коля может
послать программе
два вида запросов: строку, содержащую имя контакта и его номер,
разделенные пробелом,
```

или просто имя контакта. В первом случае программа должна добавить в книгу новый номер, во втором – вывести номер контакта. Ввод происходит до символа точки. Если введенное имя уже содержится в списке контактов, необходимо перезаписать номер.

Sample Input:

Ben 89001234050

Alice 210-220

Alice

Alice 404-502

Nick +16507811251

Ben

Alex +4(908)273-22-42

Alice

Nick

Robert 51234047129

Alex

.

Sample Output:

210-220

89001234050

404-502

+16507811251

+4(908)273-22-42

'''

d = {

}

d2 = {

}

i = 0

s = input()

lst = []

while (s):

 if (s == '.'):

 break

 s = s.split()

 if (len(s) > 1):

 d[s[0]] = s[1]

 else:

 lst.append(d[s[0]])

 s = input()

print(*lst)

Листинг 28.Mod. Телефонная Книга .

'''

<https://stepik.org/lesson/243394/step/8?unit=215740>

Телефонная книга. Этап 2. Коля понял, что у многих из его знакомых есть несколько телефонных номеров и нельзя хранить только один из них. Он попросил доработать Вашу программу так, чтобы можно было добавлять к существующему контакту новый номер или даже несколько номеров, которые передаются через запятую. По запросу телефонного номера должен выводиться весь список номеров в порядке добавления, номера должны разделяться запятой. Если у контакта нет телефонных номеров, должна выводиться строка "Не найдено".

Sample Input:

Ben 89001234050, +70504321009
Alice 210-220
Alice
Alice 404-502, 894-005, 439-095
Nick +16507811251
Ben
Alex +4(908)273-22-42
Alice
Nick
Robert 51234047129, 92174043215
Alex
Robert
.

Sample Output:

210-220
89001234050, +70504321009
210-220, 404-502, 894-005, 439-095
+16507811251
+4(908)273-22-42
51234047129, 92174043215
'''

d = {

}

i = 0

```

s = input()

lst = []

while (s):
    numbers = []
    if (s == '.'):
        break
    s = s.split()
    if (len(s) > 1):
        if (d.get(s[0]) == None):
            for i in range(1, len(s)):
                numbers.append(s[i])

            d[s[0]] = numbers
        else:
            for i in range(1, len(s)):
                d[s[0]].append(s[i])

    else:
        if d.get(s[0]) == None:

            print('\n' , "Не найдено")

            # lst.append("Не найдено")
        else:

            print('\n', *d[s[0]])

            # lst.append(d[s[0]])
    s = input()

```

1.11 Техника работы с множествами

Листинг 29.set

```
'''
```

https://pythontutor.ru/lessons/sets/problems/number_of_unique/

Задача «Количество различных чисел»

Условие. Дан список чисел. Определите, сколько в нем встречается различных чисел.

```
'''
```

```
print(len(set([int(i) for i in input().split()])))
```

Листинг 30.set2

```
'''
```

https://pythontutor.ru/lessons/sets/problems/number_of_coincidental/

Задача «Количество совпадающих чисел»

Условие. Даны два списка чисел. Посчитайте, сколько чисел содержится одновременно как в первом списке, так и во втором.

5 4 3 2 1

3 1 4 2 5

```
'''
```

```
st = set([int(i) for i in input().split()])
st2 = set([int(i) for i in input().split()])
print(len(st.intersection(st2)))
```

Листинг 31.set3

```
'''
```

https://pythontutor.ru/lessons/sets/problems/number_of_words/

Задача «Количество слов в тексте»

Условие. Дан текст: в первой строке записано число строк, далее идут сами строки.

Определите, сколько различных слов содержится в этом тексте.

Словом считается последовательность непробельных символов идущих подряд, слова разделены

одним или большим числом пробелов или символами конца строки.

```
'''
```

```
res = []
for _ in range(int(input())):
    res += input().split()
print(len(set(res)))
```


1.12 Техника работы с кортежами

Листинг 32.tuple1

```
'''
Задание 1. https://stepik.org/lesson/193753/step/4?unit=168148
Вывести чётные
Необходимо вывести все четные числа на отрезке [a; a * 10].
Sample Input:
2
Sample Output:
(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
'''

a = int(input())
if a % 2 == 0:
    print(tuple(range(a, a*10+1, 2)))
else:
    print(tuple(range(a+1, a*10+1, 2)))
```

Листинг 33.tuple2

```
'''
Задание 2. https://stepik.org/lesson/193753/step/5?unit=168148
Убывающий ряд.
С клавиатуры вводятся целые числа a > b. Выведите убывающую
последовательность чисел
по одному числу в строке.
Sample Input:
-2
-8
Sample Output:
-2
-3
-4
-5
-6
-7
'''

print( *tuple(range(int(input()), int(input()), -1)), sep='\n' )
```

Листинг 34.NamedTuple.

```
'''
Класс namedtuple() модуля collections в Python.
https://docs-python.ru/standart-library/modul-collections-  
python/klass-namedtuple-modulja-collections/
'''
```

```

'''

from collections import namedtuple

Range = namedtuple('Range', ['first', 'second'])
                                # создаем с позиционным или
именованным параметром
d = Range(int(input()), int(input()))
                                # можно обращаться по индексу
                                # как к обычному кортежу

print(d[1] - d[0])

F, S = d

print(F - S)                    # Распаковка .

F, S = d._replace(first = 0)    # Не запоминает в кортеже .

print(F - S)                    # -10
print(d.first - d.second)       # -5

print(d)                        # Range(first=5, second=10)

d = {'from': int(input()), 'to': int(input())}

print(d)

```

1.13 Техника работы с файлами

Листинг 35.File1.

```
'''
http://ptaskbook.com/ru/tasks/text.php
Text5. Дана строка S и текстовый файл. Добавить строку S в конец
файла.
'''
```

```
string = input()

with open('file.txt', 'a') as file:
    file.write('\n' + string)
    file.close()
```

Листинг 36.

```
'''
http://ptaskbook.com/ru/tasks/text.php
Text12. Дана строка S и текстовый файл. Заменить в файле все
пустые строки на строку S.
'''
```

```
lst = []
string = input()

with open('file.txt', 'r') as file:
    for line in file:
        s = string+'\n' if line == '\n' else line
        lst.append(s)
    file.close()

with open('file2.txt', 'w') as file:
    for i in range(len(lst)):
        file.write(str(lst[i]))
    file.close()

with open('file2.txt', 'r') as file:
    for line in file:
        print(line,end='')
    file.close()
```

1.14 Техника работы с модулями

Листинг 37.Collections.Deque.

```
'''
Контейнерные типы данных модуля collections.
https://docs-python.ru/standart-library/modul-collections-python/
Класс deque() модуля collections в Python.
https://docs-python.ru/standart-library/modul-collections-python/klass-deque-modulja-collections/
'''

from collections import ChainMap

# d = collections.ChainMap(*maps) # maps - один/несколько
# словарь/ей .

# Класс ChainMap() модуля collections группирует несколько
# словарей или других сопоставлений для создания
# единого обновляемого представления. Если словари maps не
# указаны, то будет создан один пустой словарь.

d = {
    'two': 22, 'three': 3
}

d2 = {
    'one': 1, 'two': 2
}

dict = ChainMap(d, d2)

print(dict) # ChainMap({'two': 22, 'three': 3}, {'one': 1,
'two': 2})

dict = ChainMap({'Year': int(input())}, {'Month': input()})

print(dict)
```

Листинг 38.Collections.Counter.

```
'''
Контейнерные типы данных модуля collections.
Класс Counter() модуля collections в Python.
```

```
https://docs-python.ru/standart-library/modul-collections-  
python/klass-counter-modulja-collections/  
'''
```

```
from collections import Counter
```

```
# класс collections.Counter() предназначен для удобных  
быстрых подсчетов
```

```
# количества появлений неизменяемых элементов в  
последовательностях.
```

```
count = Counter([1, 2, 3, 4, 1, 3, 1, 2])
```

```
dict(count)
```

```
print(count) # Counter({1: 3, 2: 2, 3: 2,  
4: 1})
```

```
# Класс Counter() модуля collections - это подкласс словаря  
dict для подсчета хеш-объектов
```

```
# (неизменяемых, таких как строки, числа, кортежи и т.д.). Это  
коллекция, в которой элементы хранятся
```

```
# в виде словарных ключей, а их счетчики хранятся в виде  
значений словаря.
```

```
print(sorted(count.elements())) # [1, 1, 1, 2, 2, 3, 3, 4]
```

```
count.update('100')
```

```
print(count)
```

Листинг 39. Collections.DefaultDict.

```
'''
```

```
Класс defaultdict() модуля collections в Python.
```

```
https://docs-python.ru/standart-library/modul-collections-  
python/klass-defaultdict-modulja-collections/  
'''
```

```
from collections import defaultdict
```

```
'''
```

```
Класс defaultdict() модуля collections возвращает новый словарь-  
подобный объект.
```

```
Defaultdict является подклассом встроенного класса dict(). Он  
переопределяет один метод и
```

добавляет одну доступную для записи переменную экземпляра. Остальная функциональность такая же, как и для класса `dict()`, и здесь она не описана.

Первый аргумент предоставляет начальное значение для атрибута `default_factory`.

По умолчанию `None`. Все остальные аргументы обрабатываются так же,

как если бы они были переданы конструктору `dict()`, включая ключевые аргументы.

```
'''
```

```
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4),
      ('red', 1)]
d = defaultdict(list)
for k, v in s:
    d[k].append(v)

print(sorted(d.items()))      #  [('blue', [2, 4]), ('red', [1]),
                                ('yellow', [1, 3])]
```

```
s = [('yellow', 1), ('blue', 2), ('yellow', 3), ('blue', 4),
      ('red', 1)]
d = {}
for k, v in s:
    d.setdefault(k, []).append(v)

print(d)                      #  {'yellow': [1, 3], 'blue': [2,
                                4], 'red': [1]}
```

```
s = 'mississippi'
d = defaultdict(int)
for k in s:
    d[k] += 1
```

```
print(sorted(d.items()))      #  [('i', 4), ('m', 1), ('p', 2),
                                ('s', 4)]
```

Листинг 40. Deque.

```
'''
```

Класс `deque()` модуля `collections` в Python.

<https://docs-python.ru/standart-library/modul-collections-python/klass-deque-modulja-collections/>

```
'''
```

```

from collections import deque
dq = deque('ghi')

'''
Класс deque() модуля collections возвращает новый объект
deque(),
инициализированный слева направо данными из итерируемой
последовательности iterable.

При создании объекта очереди класс использует метод dq.append()
для добавления элементов
из итерации iterable. Если итерация не указана, новая очередь
deque() будет пуста.
'''

print(dq)    # deque(['g', 'h', 'i'])
print(*dq)   # g h i

dq.append('. Hello!')

print(dq)    # deque(['g', 'h', 'i', '. Hello!'])
print(*dq)   # g h i . Hello!

dq.popleft()
dq.popleft()
dq.popleft()

print(dq)    # deque(['. Hello!'])

print(dq.count('. Hello!'))
# 1

dq.remove('. Hello!')

print(dq)

for i in range(3):
    dq.append(input())

print(dq)

dq.clear()

print(dq)    # deque([])

```

Листинг 41. Collections.OrderDict.

```
'''
Класс OrderedDict() модуля collections в Python.
https://docs-python.ru/standart-library/modul-collections-
python/klass-orderreddict-modulja-collections/
'''

from collections import OrderedDict

# Функция OrderedDict() модуля collections возвращает
экземпляр подкласса dict,
# у которого есть методы, специализированные для изменения
порядка словаря.
# Упорядоченные словари похожи на обычные словари, но имеют
некоторые дополнительные возможности,
# связанные с операциями упорядочивания. Теперь они стали
менее важными, когда встроенный класс dict
# получил возможность запоминать порядок вставки (это новое
поведение стало гарантированным в Python 3.7).

d = OrderedDict.fromkeys('Hello, world!')
d.move_to_end('H')
print(''.join(d.keys()))      #  elo, wrd!H

d.move_to_end('H', last=False)

print(''.join(d.keys()))      #  Helo, wrd!

d.popitem(last=True)

print(''.join(d.keys()))      #  Helo, wrd
```

Техника работы с функцией “argv” из модуля “sys” .

Листинг 42 argv.

```
import sys, os
print('Список аргументов')
print(sys.argv)
print('Исходные байты')
print([os.fsencode(arg) for arg in sys.argv])
```

Функция `argv` модуля `sys` возвращает список аргументов командной строки, передаваемых скрипту Python.

Выражение `argv[0]` - это имя скрипта и зависит от операционной системы, является ли это полный путь или нет.

Если команда была выполнена с помощью параметра командной строки `-c` для интерпретатора, то `argv[0]` устанавливает строку `'-c'`. Если имя скрипта не было передано интерпретатору Python, то `argv[0]` - это пустая строка.

Чтобы перебрать стандартный ввод или список файлов, приведенных в командной строке, смотрите модуль `fileinput`.

Запустим файл `argv.py`:

```
$ python3 argv.py -file test.txt -pi 3.14
Список параметров командной строки
['argv.py', '-file', 'test.txt', '-pi', '3.14']
Исходные байты
[b'argv.py', b'-file', b'test.txt', b'-pi', b'3.14']
```

Техника работы с sys.platform .

sys.platform:

Функция `sys.platform` возвращает строку, которая содержит идентификатор платформы, который можно использовать, например, для добавления компонентов, специфичных для платформы.

Листинг 43. Sys.Platform .

```
if sys.platform.startswith('freebsd'):
    # FreeBSD-specific code here...
elif sys.platform.startswith('linux'):
    # Linux-specific code here...
elif sys.platform.startswith('aix'):
    # AIX-specific code here...
```

Значения для других систем:

```
AIX - 'aix'
Linux - 'linux'
Windows - 'win32'
Windows/Cygwin - 'cygwin'
Macos - 'darwin'.
```

Возвращаемые именованные элементы:

- major,
- minor,
- build,
- platform,
- service_pack,
- servicepackminor,
- servicepackmajor,
- suite_mask,
- product_type
- platform_version.

К возвращаемым элементам можно получить доступ по имени, так что `sys.getwindowsversion()[0]` будет эквивалентно `sys.getwindowsversion().major`.

Для обеспечения совместимости с предыдущими версиями только первые 5 элементов могут быть извлечены при помощи индексов.

`product_type` может принимать одно из следующих значений:

1. `(VER_NT_WORKSTATION)` - Система является рабочей станцией.
2. `(VER_NT_DOMAIN_CONTROLLER)` - Система является контроллером домена.
3. `(VER_NT_SERVER)` - Система является сервером, а не контроллером домена.

`platform_version` возвращает точную основную версию, вспомогательную версию и номер сборки текущей операционной системы, а не версию, которая эмулируется для процесса.

Техника работы с версиями Python .

`sys.copyright`:

Переменная `sys.copyright` возвращает строку, содержащая авторские права, относящиеся к интерпретатору Python.

`sys.version`:

Переменная `sys.version` возвращает строку, содержащую номер версии интерпретатора Python плюс дополнительную информацию о номере сборки и используемом компиляторе.

Данная строка отображается при запуске интерактивного переводчика. Не извлекайте информацию из нее, лучше используйте `sys.version_info` и функции, предоставляемые модулем `platform`.

`sys.api_version`:

Переменная `sys.api_version` возвращает версию API языка C для этого интерпретатора Python. Программисты могут использовать ее при отладке конфликтов версий между Python и модулями расширения.

sys.version_info:

Переменная `sys.api_version` возвращает кортеж, содержащий пять компонентов номера версии:

- `major`,
- `minor`,
- `micro`,
- `releaselevel`,
- `serial`.

Все значения, кроме `releaselevel`, являются целыми числами.

Значения `releaselevel`:

- `'alpha'`,
- `'beta'`,
- `'candidate'`
- `'final'`.

Компоненты также могут быть доступны по имени, `sys.version_info[0]` что эквивалентно `sys.version_info.major` и так далее.

sys.implementation:

Атрибут `sys.implementation` представляет собой объект, содержащий информацию о реализации текущего запущенного интерпретатора Python. Следующие атрибуты должны существовать во всех реализациях Python.

`name` - это идентификатор реализации, например `"cpython"`. Фактическая строка определяется реализацией Python, но она гарантированно будет строчной.

`version` - это именованный кортеж, в том же формате, что и `sys.version_info`. Он представляет собой версию реализации Python. Значение `version` отличается от конкретной версии языка Python, которой соответствует текущий работающий интерпретатор. Например, для PyPy 1.8 `sys.implementation.version` может быть `sys.version_info (1, 8, 0, 'final', 0)`,

тогда как `sys.version_info` будет `sys.version_info (2, 7, 2, 'final', 0)`. Для CPython они имеют одинаковое значение, так как это эталонная реализация.

`hexversion` - это версия реализации в шестнадцатеричном формате, например `sys.hexversion`.

`cache_tag` - это тег, используемый механизмом импорта в именах файлов кэшируемых модулей. По соглашению, это будет составная часть имени и версии реализации, например, "cpython-38". Хотя реализация Python может использовать другое значение, если это уместно. Если для `cache_tag` установлено значение `None`, это означает, что кэширование модуля должно быть отключено.

Атрибут `sys.implementation` может содержать дополнительные атрибуты, специфичные для реализации Python. Эти нестандартные атрибуты должны начинаться с подчеркивания и здесь не описаны. Независимо от содержимого `sys.implementation` не изменится ни во время выполнения интерпретатора, ни между версиями реализации. Однако, может измениться между версиями языка Python.

`sys.hexversion`:

Значение `sys.hexversion` содержит номер версии, закодированный как одно целое число. Оно гарантированно увеличивается с каждой версией, включая надлежащую поддержку выпусков. Например, чтобы проверить, что интерпретатор Python имеет версию не ниже 3.5.2, используйте:

Листинг 44. `sys.HexVersion`.

```
if sys.hexversion >= 0x030502F0:
    # используйте дополнительные функции ...
else:
    # используйте альтернативную реализацию # или предупредите
    пользователя ...
```

Значение `sys.hexversion` это шестнадцатеричный номер версии и представляет собой результат передачи его встроенной функции `hex()`. Для

более удобного получения той же самой информации может использоваться именованный кортеж, возвращаемый `sys.version_info`.

`sys.winver`:

Номер версии, используемый для формирования ключей реестра на платформах Windows. Значение хранится как строковый ресурс 1000 в DLL-библиотеке Python. Обычно это первые три символа, возвращаемые `sys.version`. Значение `sys.winver` представлен в модуле `sys` в ознакомительных целях, изменение этого значения не влияет на ключи реестра, используемые Python.

Техника работы с каталогами и путями интерпретатора Python .

sys.prefix:

Атрибут `sys.prefix` это строка, предоставляет специфичный для площадки префикс каталога, в котором установлены независимые от платформы файлы Python. По умолчанию, это строка `'/usr/local'`.

Значение `sys.prefix` устанавливается во время сборки Python с помощью аргумента `--prefix` для скрипта `configure`, например `$./configure --prefix=/opt/python-3.x.x/`. Основная коллекция модулей библиотеки Python установлена в каталоге `prefix/lib/pythonX.Y`, а независимые от платформы заголовочные файлы (все, кроме `pyconfig.h`) хранятся в `prefix/include/pythonX.Y`, где XY - номер версии Python, например 3.7.

Примечание. Если запущена виртуальная среда, это значение будет изменено в `site.py` для указания на виртуальную среду. Значение для установки Python по-прежнему будет доступно через `base_prefix`.

sys.base_prefix:

Атрибут `sys.base_prefix` устанавливается при запуске Python перед запуском `site.py` то же значение, что и атрибут `sys.prefix`. Если не работает в виртуальной среде, значения останутся прежними. Если `site.py` обнаружит, что используется виртуальная среда, то значения `sys.prefix` и `sys.exec_prefix` будут изменены на значения установки виртуальной среды, тогда как `sys.base_prefix` и `sys.base_exec_prefix` останутся указывать на базовую установку Python (ту, из которой была создана виртуальная среда).

sys.exec_prefix:

Атрибут `sys.exec_prefix` это строка, задает префикс каталога для конкретной программы, где установлены зависящие от платформы файлы Python. По умолчанию это `'/usr/local'`.

Значение `sys.prefix` устанавливается во время сборки Python с помощью аргумента `--exec-prefix` для скрипта `configure`. В частности, все файлы конфигурации (например, заголовочный файл `pyconfig.h`) установлены в каталоге `exec_prefix/lib/pythonX.Y/config`, а модули разделяемой библиотеки

установлены в `exec_prefix/lib/pythonX.Y/lib-dynload`, где `XY` номер версии Python, например 3.7.

Примечание. Если запущена виртуальная среда, то значение `sys.exec_prefix` будет изменено в `site.py` для указания на виртуальную среду. Значение базовой установки Python по-прежнему будет доступно через атрибут `sys.base_exec_prefix`.

`sys.base_exec_prefix`:

Атрибут `sys.exec_prefix` устанавливает при запуске Python до запуска `site.py` то же значение, что и `sys.exec_prefix`. Если программа не работает в виртуальной среде, то значения останутся прежними. Если `site.py` обнаружит, что используется виртуальная среда, то значения `sys.prefix` и `sys.exec_prefix` будут изменены на значения установки виртуальной среды, тогда как `sys.base_prefix` и `sys.base_exec_prefix` останутся указывать на базовую установку Python (ту, из которой была создана виртуальная среда).

`sys.executable`:

Атрибут `sys.executable` это строка, задающая абсолютный путь исполняемого двоичного файла для интерпретатора Python в системах, где это имеет смысл. Если Python не может получить реальный путь к своему исполняемому файлу, `sys.executable` будет пустой строкой или `None`.

`sys.platlibdir`:

Атрибут `sys.exec_prefix` это имя каталога библиотеки для конкретной платформы. Он используется для построения пути к стандартной библиотеке и путей установленных модулей расширения. (Новое в Python 3.9.)

На большинстве платформ он равен `lib`. В Fedora и SuSE на 64-битных платформах он равен `lib64`, что дает следующие пути `sys.path` (где `X.Y` - версия Python `major.minor`):

- `/usr/lib64/pythonX.Y/`: Стандартная библиотека (например, `os.py` модуля `os`),

- `/usr/lib64/pythonX.Y/lib-dynload/`: модули расширения языка C стандартной библиотеки (например, модуль `errno`, точное имя файла зависит от платформы),
- `/usr/lib/pythonX.Y/site-packages/` (всегда используйте `lib`, а не `sys.platlibdir`): сторонние модули,
- `/usr/lib64/pythonX.Y/site-packages/`: модули расширения языка со сторонних пакетов.

Техника работы с `sys.std` .

Файловые объекты интерпретатора для стандартного ввода, вывода.
Синтаксис:

Листинг 45. `Sys.STD` .

```
import sys
sys.stdin
sys.stdout
sys.stderr
```

Описание:

Файловые

объекты `sys.stdin`, `sys.stdout` и `sys.stderr` используются интерпретатором для стандартного ввода, вывода и ошибок:

- `sys.stdin` - используется для всех интерактивных входных данных, включая вызовы `input()`;
- `sys.stdout` - используется для вывода оператором `print()` и выражений, которые возвращают значение, а также для подсказок `input()`;
- `sys.stderr` - сообщения об ошибках и собственные запросы переводчика.

Эти потоки являются обычными текстовыми файлами, такими же, как и те, которые возвращаются функцией `open()` .

Их параметры выбираются следующим образом:

- Кодировка символов зависит от платформы. Не-Windows платформы используют кодировку локали `locale.getpreferredencoding()`.
- В Windows UTF-8 используется для консольного устройства. Не символьные устройства, такие как дисковые файлы и каналы, используют кодировку языкового стандарта системы, то есть кодовую страницу ANSI. Не консольные символьные устройства, такие как `NUL`, т.е. где метод `isatty()` возвращает `True`, используют значение кодовых страниц ввода и вывода консоли при запуске соответственно для `sys.stdin` и `sys.stdout/sys.stderr`. По умолчанию используется системная кодировка локали, если процесс изначально не подключен к консоли.

Особое поведение консоли можно изменить, установив переменную окружения `PYTHONLEGACYWINDOWSSTDIO` перед запуском Python. В этом случае кодовые страницы консоли используются как для любого другого символьного устройства.

На всех платформах можно переопределить кодировку символов, установив переменную среды `PYTHONIOENCODING` перед запуском Python или используя новый параметр командной строки - `X utf8` и переменную среды `PYTHONUTF8`. Однако для консоли Windows это применимо только в том случае, если также установлено `PYTHONLEGACYWINDOWSSTDIO`.

В интерактивном режиме потоки `stdout` и `stderr` буферизуются по строкам. В противном случае они буферизируются как обычные текстовые файлы. Вы можете переопределить это значение с помощью параметра командной строки `-u`.

Примечание. Для записи или чтения двоичных данных из/в стандартные потоки используйте базовый объект двоичного буфера. Например, чтобы

записать байты в `sys.stdout` используйте `sys.stdout.buffer.write(b'abc')`.

Однако, если вы пишете модуль или библиотеку и не контролируете, в каком контексте будет выполняться ее код, помните, что стандартные потоки могут быть заменены файловыми объектами, такими как `io.StringIO`, которые не поддерживают атрибут буфера.

Изменено в Python 3.9: не интерактивный `sys.stderr` теперь буферизуется по строкам, а не полностью.

`sys.__stdin__`,

`sys.__stdout__`,

`sys.__stderr__`:

Эти объекты содержат исходные значения `sys.stdin`, `sys.stderr` и `sys.stdout`. Они используются во время финализации и могут быть полезны для печати в реальном стандартном потоке, независимо от того, был ли перенаправлен объект `sys.std*`.

Его также можно использовать для восстановления реальных файлов в известные рабочие файловые объекты, если они были перезаписаны поврежденным объектом. Однако предпочтительный способ сделать это - явно сохранить предыдущий поток перед его заменой и восстановить сохраненный объект.

Примечание. При некоторых условиях `sys.stdin`, `sys.stderr` и `sys.stdout`, а также исходные значения `sys.__stdin__`, `sys.__stdout__` и `sys.__stderr__` могут быть `None`. Обычно это относится к приложениям с графическим интерфейсом Windows, которые не подключены к консоли, а приложения Python запускаются с `pythonw.exe`⁽¹⁾.

Примеры использования `sys.stdout`, `sys.stdin`:

- Запись в файл консольного вывода;
- Печать в консоль с задержкой (как телетекст...).

Запись в файл того, что выводится на консоли (лог консоли).

Этот код ничего не выводит на консоль, но записывает "Hello World" в текстовый файл с именем `file.txt`.

Листинг 46. Sys.STD + File.txt .

```
stdout = sys.stdout
try:
    sys.stdout = open('file.txt', 'w')
    print('Hello World')
finally:
    # Закрываем file.txt
    sys.stdout.close()
    sys.stdout = stdout
```

Техника работы с функцией `sys.exit()` .

Листинг 47. `sys.Exit` .

```
import sys

sys.exit([arg])
```

Функция `exit()` модуля `sys` - выход из Python. Она реализуется путем вызова исключения `SystemExit`, поэтому выполняются действия по очистке, указанные в предложениях `finally` операторов `try` и можно перехватить попытку выхода на внешнем уровне.

Необязательный аргумент `arg` может быть целым числом, указывающим статус выхода (по умолчанию равен нулю) или другим типом объекта. Если это целое число, ноль считается "успешным завершением", а любое ненулевое значение считается "ненормальным завершением" и т.п.

Большинство систем требуют, чтобы `arg` находился в диапазоне 0 - 127, и в противном случае дают неопределенные результаты. Некоторые системы имеют соглашение о назначении определенных значений определенным кодам выхода, но они, как правило, недостаточно развиты. Программы Unix обычно используют 2 для ошибок синтаксиса командной строки и 1 для всех других видов ошибок. Если передается объект другого типа, то `None` эквивалентен передаче нуля, а любой другой объект выводится на `sys.stderr` и приводит к коду выхода 1. В частности, `sys.exit()` - это быстрый способ выйти из программы при возникновении ошибки.

Техника работы с файлами с помощью модуля `OS`.

Обработка файлов в Python с помощью модуля `os` включает создание, переименование, перемещение, удаление файлов и папок, а также получение списка всех файлов и каталогов и многое другое.

В индустрии программного обеспечения большинство программ тем или иным образом обрабатывают файлы: создают их, переименовывают, перемещают и так далее. Любой программист должен обладать таким навыком. С этим руководством вы научитесь использовать модуль `os` в Python для проведения операций над файлами и каталогами вне зависимости от используемой операционной системы.

Важно знать, что модуль `os` используется не только для работы с файлами. Он включает массу методов и инструментов для других операций: обработки переменных среды, управления системными процессами, а также аргументы командной строки и даже расширенные атрибуты файлов, которые есть только в Linux.

Модуль встроенный, поэтому для работы с ним не нужно ничего устанавливать.

Вывод текущей директории

Для получения текущего рабочего каталога используется `os.getcwd()`:

Листинг 48. `OS.GetCWD` .

```
import os
# вывести текущую директорию
print("Текущая деректория:", os.getcwd())
```

`os.getcwd()` возвращает строку в Юникоде, представляющую текущий рабочий каталог. Вот пример вывода:

Текущая деректория: C:\python3\bin

Создание папки

Для создания папки/каталога в любой операционной системе нужна следующая команда:

Листинг 49. `OS.MKDir`.

```
# создать пустой каталог (папку)
os.mkdir("folder")
```

После ее выполнения в текущем рабочем каталоге тут же появится новая папка с названием «folder».

Если запустить ее еще раз, будет вызвана ошибка `FileExistsError`, потому что такая папка уже есть. Для решения проблемы нужно запускать команду только в том случае, если каталога с таким же именем нет. Этого можно добиться следующим образом:

Листинг 50. `FileExistsError`.

```
# повторный запуск mkdir с тем же именем вызывает
FileExistsError,
# вместо этого запустите:
if not os.path.isdir("folder"):
    os.mkdir("folder")
```

Функция `os.path.isdir()` вернет `True`, если переданное имя ссылается на существующий каталог.

Изменение директории

Менять директории довольно просто. Проделаем это с только что созданным:

Листинг 51. `OS.CHDir`.

```
# изменение текущего каталога на 'folder'
os.chdir("folder")
```

Еще раз выведем рабочий каталог:

```
# вывод текущей папки
print("Текущая директория изменилась на folder:", os.getcwd())
```

Вывод:

Текущая директория изменилась на folder: `C:\python3\bin\folder`

Создание вложенных папок

Предположим, вы хотите создать не только одну папку, но и несколько вложенных:

Листинг 52.OS.MakeDirs.

```
# вернуться в предыдущую директорию
os.chdir("..")

# сделать несколько вложенных папок
os.makedirs("nested1/nested2/nested3")
```

Это создаст три папки рекурсивно, как показано на следующем изображении:

Создание файлов

Для создания файлов в Python модули не нужны. Можно использовать встроенную функцию `open()`. Она принимает название файла, который необходимо создать в качестве первого параметра и желаемый режим открытия — как второй:

Листинг 53.File.Open.

```
# создать новый текстовый файл
text_file = open("text.txt", "w")
# записать текста в этот файл
text_file.write("Это текстовый файл")
```

`w` значит `write` (запись), а `a` — это `appending` (добавление данных к уже существующему файлу), а `r` — `reading` (чтение). Больше о режимах открытия можно почитать здесь.

Переименование файлов

С помощью модуля `os` достаточно просто переименовать файл. Поменяем название созданного в прошлом шаге.

Листинг 54.OS.Rename.

```
# переименовать text.txt на renamed-text.txt
os.rename("text.txt", "renamed-text.txt")
```

Функция `os.rename()` принимает 2 аргумента: имя файла или папки, которые нужно переименовать и новое имя.

Перемещение файлов

Функцию `os.replace()` можно использовать для перемещения файлов или каталогов:

Листинг 55.OS.Replace.

```
# заменить (переместить) этот файл в другой каталог
os.replace("renamed-text.txt", "folder/renamed-text.txt")
```

Стоит обратить внимание, что это перезапишет путь, поэтому если в папке `folder` уже есть файл с таким же именем (`renamed-text.txt`), он будет перезаписан.

Список файлов и директорий

Листинг 56.OS.ListDir.

```
# распечатать все файлы и папки в текущем каталоге
print("Все папки и файлы:", os.listdir())
```

Функция `os.listdir()` возвращает список, который содержит имена файлов в папке. Если в качестве аргумента не указывать ничего, вернется список файлов и папок текущего рабочего каталога:

Все папки и файлы: `['folder', 'handling-files', 'nested1', 'text.txt']`

А что если нужно узнать состав и этих папок тоже? Для этого нужно использовать функцию `os.walk()`:

Листинг 57.OS.Walk.

```
# распечатать все файлы и папки рекурсивно
for dirpath, dirnames, filenames in os.walk("."):
    # перебрать каталоги
    for dirname in dirnames:
        print("Каталог:", os.path.join(dirpath, dirname))
    # перебрать файлы
    for filename in filenames:
        print("Файл:", os.path.join(dirpath, filename))
```

`os.walk()` — это генератор дерева каталогов. Он будет перебирать все переданные составляющие. Здесь в качестве аргумента передано значение «.», которое обозначает верхушку дерева:

Каталог: .\folder

Каталог: .\handling-files

Каталог: .\nested1

Файл: .\text.txt

Файл: .\handling-files\listing_files.py

Файл: .\handling-files\README.md

Каталог: .\nested1\nested2

Каталог: .\nested1\nested2\nested3

Метод `os.path.join()` был использован для объединения текущего пути с именем файла/папки.

Удаление файлов

Удалим созданный файл:

Листинг 58.OS.Remove.

```
# удалить этот файл
os.remove("folder/renamed-text.txt")
```

`os.remove()` удалит файл с указанным именем (не каталог).

Удаление директорий

С помощью функции `os.rmdir()` можно удалить указанную папку:

```
# удалить папку
os.rmdir("folder")
```

Для удаления каталогов рекурсивно необходимо использовать `os.removedirs()`:

```
# удалить вложенные папки
os.removedirs("nested1/nested2/nested3")
```

Это удалит только пустые каталоги.

Получение информации о файлах

Для получения информации о файле в ОС используется функция `os.stat()`, которая выполняет системный вызов `stat()` по выбранному пути:

Листинг 59.OS.Stat.

```
open("text.txt", "w").write("Это текстовый файл")

# вывести некоторые данные о файле
print(os.stat("text.txt"))
```

Вывод:

```
os.stat_result(st_mode=33206, st_ino=14355223812608232, st_dev=1558443184,
st_nlink=1, st_uid=0, st_gid=0, st_size=19, st_atime=1575967618,
st_mtime=1575967618, st_ctime=1575966941)
```

Это вернет кортеж с отдельными метриками. В их числе есть следующие:

- `st_size` — размер файла в байтах
- `st_atime` — время последнего доступа в секундах (временная метка)
- `st_mtime` — время последнего изменения

- `st_ctime` — в Windows это время создания файла, а в Linux — последнего изменения метаданных

Для получения конкретного атрибута нужно писать следующим образом:

Листинг 60.`os.stat().st_size`.

```
# например, получить размер файла
print("Размер файла:", os.stat("text.txt").st_size)
```

Вывод:

Размер файла: 19 .

1.15 Техника работы с классами

Создание класса

Для того, чтобы создать класс, используйте ключевое слово `class`.

Создадим класс с именем `MyClass` и свойством `x`:

Листинг 61.class1

```
class MyClass:
    x = 5
```

Создание объекта

Теперь мы можем использовать класс под названием `myClass` для создания объектов.

Создадим объект под названием `p1`, и выведем значение `x`:

Листинг 62.print(class.attribute).

```
p1 = MyClass()
print(p1.x)
```

Вывод:

5

Функция `__init__`

Приведенные выше примеры — это классы и объекты в их простейшей форме и не очень полезны в приложениях.

Чтобы понять значение классов, нам нужно понять встроенную функцию `__init__`.

У всех классов есть функция под названием `__init__()`, которая всегда выполняется при создании объекта. Используйте функцию `__init__()` для добавления значений свойствам объекта или других операций, которые необходимо выполнить, при создании объекта.

Для создания класса под названием `Person`, воспользуемся функцией `__init__()`, что бы добавить значения для имени и возраста: *class Person:*

Листинг 64.`def __init__()`.

```
def __init__(self, name, age):
#     name    name of object
#     age     how old is object
    self.name = name
    self.age = age
p1 = Person("Василий", 36)
print(p1.name)
print(p1.age)
```

Вывод:

Василий

36

Обратите внимание: Функция `__init__()` автоматически вызывается каждый раз при использовании класса для создания нового объекта.

Методы объектов

Объекты также содержат методы. Методы в объектах — это функции, принадлежащие объекту.

Давайте создадим метод в классе Person.

Добавим функцию, которая выводит приветствие, и выполним ее:

Листинг 65.`def myfunc()`.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Привет, меня зовут " + self.name)

p1 = Person("Василий", 36)
p1.myfunc()
```

Вывод:

Привет, меня зовут Василий

Параметр `self`

Параметр `self` является ссылкой на сам класс и используется для доступа к переменным, принадлежащим классу.

Его не обязательно называть `self`, вы можете называть его как хотите, но он должен быть первым параметром любой функции в классе.

Используем слова `mysillyobject` и `abc` вместо `self`:

Листинг 66.`self` меняем на `mysillyobject` и `abc`.

```
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age
    def myfunc(abc):
        print("Привет, меня зовут " + abc.name)
p1 = Person("Василий", 36)
p1.myfunc()
```

Вывод:

Привет, меня зовут Василий

Изменение свойств объекта

Вы можете изменять свойства объектов следующим образом.

Изменим возраст от p1 на 40:

Листинг 67. Присвоение значения переменной age из атрибута p1.

```
p1.age = 40
```

Больше примеров применения class в Python 3: Примеры работы с классами в Python

Удалить свойства объекта

Свойства объектов можно удалять с помощью ключевого слова del

Листинг 68. Удаление переменной age из атрибута p1.

```
del p1.age
```

Удаление объектов

Вы можете удалить объекты, используя ключевое слово del.

Создание классов

Оператор class создает новое определение класса. Имя класса сразу следует за ключевым словом class, после которого ставится двоеточие:

Листинг 69.class Name.

```
class ClassName:
    """Необязательная строка документации класса"""
    class_suite
```

- У класса есть строка документации, к которой можно получить доступ через ClassName.__doc__.
- class_suite состоит из частей класса, атрибутов данных и функций.

Пример создания класса

Создание простого класса на Python

Листинг 70.class.

```
class Employee:
    """Базовый класс для всех сотрудников"""
    emp_count = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.emp_count += 1
    def display_count(self):
        print('Всего сотрудников: %d' % Employee.empCount)

    def display_employee(self):
        print('Имя: {}. Зарплата: {}'.format(self.name,
self.salary))
```

- Переменная `emp_count` — переменная класса, значение которой разделяется между экземплярами этого класса. Получить доступ к этой переменной можно через `Employee.emp_count` из класса или за его пределами.
- Первый метод `__init__()` — специальный метод, который называют конструктором класса или методом инициализации. Его вызывает Python при создании нового экземпляра этого класса.
- Объявляйте другие методы класса, как обычные функции, за исключением того, что первый аргумент для каждого метода `self`. Python добавляет аргумент `self` в список для вас; и тогда вам не нужно включать его при вызове этих методов.

Создание экземпляров класса

Чтобы создать экземпляры классов, нужно вызвать класс с использованием его имени и передать аргументы, которые принимает метод `__init__`.

Листинг 71. Attribute using `__init__()`.

```
# Это создаст первый объект класса Employee
emp1 = Employee("Андрей", 2000)
# Это создаст второй объект класса Employee
emp2 = Employee("Мария", 5000)
```

Доступ к атрибутам

Получите доступ к атрибутам класса, используя оператор `.` после объекта класса. Доступ к классу можно получить используя имя переменной класса:

Листинг 72. Доступ к классу.

```
emp1.display_employee()
emp2.display_employee()
print("Всего сотрудников: %d" % Employee.emp_count)
```

Теперь, систематизируем все.

Листинг 73.

```
class Employee:
    """Базовый класс для всех сотрудников"""
    emp_count = 0

    def __init__(self, name, salary):
        # name Object's name.
        # salary ЗП объекта . Object's salary.
        self.name = name
        self.salary = salary
        Employee.emp_count += 1

    def display_count(self):
        print('Всего сотрудников: %d' % Employee.emp_count)

    def display_employee(self):
        print('Имя: {}. Зарплата: {}'.format(self.name,
self.salary))
```

```
# Это создаст первый объект класса Employee
emp1 = Employee("Андрей", 2000)
# Это создаст второй объект класса Employee
emp2 = Employee("Мария", 5000)
emp1.display_employee()
emp2.display_employee()
print("Всего сотрудников: %d" % Employee.emp_count)
```

При выполнении этого кода, мы получаем следующий результат:

Имя: Андрей. Зарплата: 2000

Имя: Мария. Зарплата: 5000

Всего сотрудников: 2

Вы можете добавлять, удалять или изменять атрибуты классов и объектов в любой момент.

```
emp1.age = 7 # Добавит атрибут 'age'.
emp1.age = 8 # Изменит атрибут 'age'.
del emp1.age # Удалит атрибут 'age'.
```

Вместо использования привычных операторов для доступа к атрибутам вы можете использовать эти функции:

- `getattr(obj, name [, default])` — для доступа к атрибуту объекта.
- `hasattr(obj, name)` — проверить, есть ли в `obj` атрибут `name`.
- `setattr(obj, name, value)` — задать атрибут. Если атрибут не существует, он будет создан.
- `delattr(obj, name)` — удалить атрибут

```
hasattr(emp1, 'age') # возвращает true если атрибут 'age'
                     # существует
getattr(emp1, 'age') # возвращает значение атрибута 'age'
setattr(emp1, 'age', 8) # устанавливает атрибут 'age' на 8
delattr(emp1, 'age') # удаляет атрибут 'age'
```

Встроенные атрибуты класса

Каждый класс Python хранит встроенные атрибуты, и предоставляет к ним доступ через оператор `.`, как и любой другой атрибут

- `__dict__` — словарь, содержащий пространство имен класса.
- `__doc__` — строка документации класса. `None` если, документация отсутствует.
- `__name__` — имя класса.
- `__module__` — имя модуля, в котором определяется класс. Этот атрибут `__main__` в интерактивном режиме.
- `__bases__` — могут быть пустые tuple, содержащие базовые классы, в порядке их появления в списке базового класса.

Для вышеуказанного класса давайте попробуем получить доступ ко всем этим атрибутам:

Листинг 74. Доступ ко всем атрибутам.

```
class Employee:
    """Базовый класс для всех сотрудников"""
    emp_count = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def display_count(self):
        print('Всего сотрудников: %d' % Employee.empCount)

    def display_employee(self):
        print('Имя: {}. Зарплата: {}'.format(self.name,
self.salary))

print("Employee.__doc__:", Employee.__doc__)
print("Employee.__name__:", Employee.__name__)
print("Employee.__module__:", Employee.__module__)
print("Employee.__bases__:", Employee.__bases__)
print("Employee.__dict__:", Employee.__dict__)
```

Когда этот код выполняется, он возвращает такой результат:

`Employee.__doc__`: Базовый класс для всех сотрудников

Employee.__name__: Employee

Employee.__module__: __main__

Employee.__bases__: (<class 'object'>,)

Employee.__dict__: {'__module__': '__main__', '__doc__': 'Базовый класс для всех сотрудников', 'emp_count': 0, '__init__': <function Employee.__init__ at 0x03C7D7C8>, 'display_count': <function Employee.display_count at 0x03FA6AE0>, 'display_employee': <function Employee.display_employee at 0x03FA6B28>, '__dict__': <attribute '__dict__' of 'Employee' objects>, '__weakref__': <attribute '__weakref__' of 'Employee' objects>}

Уничтожение объектов (сбор мусора)

Python автоматически удаляет ненужные объекты (встроенные типы или экземпляры классов), чтобы освободить пространство памяти. С помощью процесса ‘Garbage Collection’ Python периодически восстанавливает блоки памяти, которые больше не используются.

Сборщик мусора Python запускается во время выполнения программы и тогда, когда количество ссылок на объект достигает нуля. С изменением количества обращений к нему, меняется количество ссылок.

Когда объект присваивают новой переменной или добавляют в контейнер (список, кортеж, словарь), количество ссылок объекта увеличивается. Количество ссылок на объект уменьшается, когда он удаляется с помощью del, или его ссылка выходит за пределы видимости.

Когда количество ссылок достигает нуля, Python автоматически собирает его.

Листинг 75.delete.

```
a = 40          # создали объект <40>
b = a          # увеличивает количество ссылок <40>
c = [b]         # увеличивает количество ссылок <40>

del a          # уменьшает количество ссылок <40>
b = 100        # уменьшает количество ссылок <40>
c[0] = -1      # уменьшает количество ссылок <40>
```

Обычно вы не заметите, когда сборщик мусора уничтожает экземпляр и очищает свое пространство. Но классом можно реализовать специальный метод `__del__()`, называемый деструктором. Он вызывается, перед уничтожением экземпляра. Этот метод может использоваться для очистки любых ресурсов памяти.

Пример работы `__del__()`

Деструктор `__del__()` выводит имя класса того экземпляра, который должен быть уничтожен

Листинг 76. Деструктор (`__del__()`).

```
class Point:
    def __init__(self, x=0, y=0):
#       x and y are coordinates of Point's attribute .
        self.x = x
        self.y = y

    def __del__(self):
        class_name = self.__class__.__name__
        print('{} уничтожен'.format(class_name))

pt1 = Point()
pt2 = pt1
pt3 = pt1
print(id(pt1), id(pt2), id(pt3)) # выведите id объектов
del pt1
del pt2
del pt3
```

Когда вышеуказанный код выполняется и выводит следующее:

```
17692784 17692784 17692784
```

Point уничтожен

В идеале вы должны создавать свои классы в отдельном модуле. Затем импортировать их в основной модуль программы с помощью `import SomeClass`.

Наследование класса

Вместо того, чтобы начинать с нуля, вы можете создать класс, на основе уже существующего. Укажите родительский класс в круглых скобках после имени нового класса.

Класс наследник наследует атрибуты своего родительского класса. Вы можете использовать эти атрибуты так, как будто они определены в классе наследнике. Он может переопределять элементы данных и методы родителя.

Синтаксис наследования класса

Классы наследники объявляются так, как и родительские классы. Только, список наследуемых классов, указан после имени класса.

Листинг 77. Класс наследник.

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    """Необязательная строка документации класса"""  
    class_suite
```

Пример наследования класса в Python

Листинг 78. Наследование.

```
class Parent: # объявляем родительский класс  
    parent_attr = 100  
  
    def __init__(self):  
        print('Вызов родительского конструктора')  
  
    def parent_method(self):  
        print('Вызов родительского метода')
```

```

    def set_attr(self, attr):
#       class's var assign to new var "attr".
        Parent.parent_attr = attr

    def get_attr(self):
        print('Атрибут родителя: {}'.format(Parent.parent_attr))

class Child(Parent): # объявляем класс наследник
    def __init__(self):
        print('Вызов конструктора класса наследника')

    def child_method(self):
        print('Вызов метода класса наследника')

c = Child() # экземпляр класса Child
c.child_method() # вызов метода child_method
c.parent_method() # вызов родительского метода parent_method
c.set_attr(200) # еще раз вызов родительского метода
c.get_attr() # снова вызов родительского метода

```

Когда этот код выполняется, он выводит следующий результат:

Вызов конструктора класса наследника

Вызов метода класса наследника

Вызов родительского метода

Атрибут родителя: 200

Аналогичным образом вы можете управлять классом с помощью нескольких родительских классов:

Листинг 79. Управление классом с помощью нескольких родительских классов.

```

class A:          # объявите класс A
    ....
class B:          # объявите класс B
    ....
class C(A, B):    # C наследуется от A и B

```


Вы можете использовать функции `issubclass()` или `isinstance()` для проверки отношений двух классов и экземпляров.

- Логическая функция `issubclass(sub, sup)` возвращает значение `True`, если данный подкласс `sub` действительно является подклассом `sup`.
- Логическая функция `isinstance(obj, Class)` возвращает `True`, если `obj` является экземпляром класса `Class` или является экземпляром подкласса класса.

Переопределение методов

Вы всегда можете переопределить методы родительского класса. В вашем подклассе могут понадобиться специальные функции. Это одна из причин переопределения родительских методов.

Пример переопределения методов

Листинг 80. Переопределение методов .

```
class Parent: # объявите родительский класс
    def my_method(self):
        print('Вызов родительского метода')
class Child(Parent): # объявите класс наследник
    def my_method(self):
        print('Вызов метода наследника')
c = Child() # экземпляр класса Child
c.my_method() # метод переопределен классом наследником
```

Когда этот код выполняется, он производит следующий результат:

Вызов метода наследника

Популярные базовые методы

В данной таблице перечислены некоторые общие функции. Вы можете переопределить их в своих собственных классах.

| № | Метод, описание и пример вызова |
|---|--|
| 1 | <code>__init__(self [, args...])</code> – конструктор (с любыми необязательными аргументами) <code>obj = className(args)</code> |
| 2 | <code>__del__(self)</code> – деструктор, удаляет объект <code>del obj</code> |
| 3 | <code>__repr__(self)</code> – оценочное строковое представление <code>repr(obj)</code> |
| 4 | <code>__str__(self)</code> – печатное строковое представление <code>str(obj)</code> |

Пример использования `__add__`

Предположим, вы создали класс `Vector` для представления двумерных векторов. Что происходит, когда вы используете дополнительный оператор для их добавления? Скорее всего, Python будет против.

Однако вы можете определить метод `__add__` в своем классе для добавления векторов и оператор `+` будет вести себя так как нужно.

Листинг 81. `def __add__()`.

```
class Vector:
    def __init__(self, a, b)
#       class's vars a and b assign to new vars a and b.
        self.a = a
        self.b = b
    def __str__(self):
        return 'Vector ({}, {})'.format(self.a, self.b)
```

```

    def __add__(self, other):
#       other other object of Vector's class .
        return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(2, 10)
v2 = Vector(5, -2)
print(v1 + v2)

```

При выполнении этого кода, мы получим:

```
Vector(7, 8)
```

Приватные методы и атрибуты класса:

Атрибуты класса могут быть не видимыми вне определения класса. Вам нужно указать атрибуты с `__` вначале, и эти атрибуты не будут вызваны вне класса.

Пример приватного атрибута

Листинг 82. Приватные атрибуты.

```

class JustCounter:
    __secret_count = 0
    def count(self):
        self.__secret_count += 1
        print(self.__secret_count)
counter = JustCounter()
counter.count()
counter.count()
print(counter.__secret_count)

```

При выполнении данного кода, имеем следующий результат:

```

Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print(counter.__secret_count)
AttributeError: 'JustCounter' object has no attribute
'__secret_count'

```

Вы можете получить доступ к таким атрибутам, так `object._className_attrName`. Если вы замените свою последнюю строку следующим образом, то она будет работать.

.....

```
print(counter._JustCounter__secretCount)
```

При выполнении кода, получаем результат:

1

2

2

Листинг 83.del attribute p1.

```
del p1
```

Листинг 84. MyClass .

```
'''
Задание 1. Придумать собственный класс
Задание 2. Неформально описать функционал класса
Задание 3. Реализовать класс в модуле
Задание 4. Разработать скрипт для демонстрации работы с классом
(импортировать модуль,
создать экземпляры, вызвать методы)
'''
class RGB:
    Red = Green = Blue = 0    #    объявление и инициализация.

    def __init__(self, r, g, b): #    конструктор класса .
#    got vars for filling Red, Green and Blue params of Color .
        RGB.Red, RGB.Green, RGB.Blue = r, g, b

    def getParams(self):        #    function getParams, returns
                                #    dict.

        d = {
            'Red: ': RGB.Red,
            'Green: ': RGB.Green,
            'Blue: ': RGB.Blue
        }

        return d

Color = RGB(int(input()), int(input()), int(input())) # attribute

print(Color.getParams()) #    вызов метода класса
                        #    RGB(getParams())
```