



- * **Ветвления.**
- * **Оператор if.**
- * **Базовая форма цикла while.**
- * **Операторы break и continue.**
- * **Перебор (for).**
- * **Генераторы словарей, списков, множеств.**



Турашова Анна Николаевна

Преподаватель

anna1turashova@gmail.com

Telegram: @anna1tur



Поверка домашнего задания

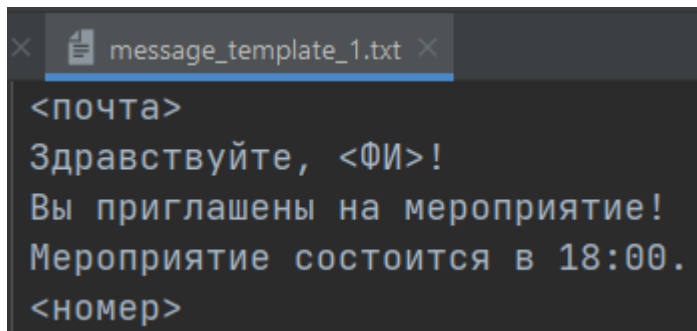


Вам дан файл-шаблон с неизвестным заранее содержимым и неизвестным количеством строк. А также файл с почтами, именами и фамилиями.

Вам необходимо написать алгоритм, который будет создавать новые файлы на основе шаблона. В новом файле на первой строке должна быть указана почта адресата, а на последней – номер файла в формате «номер_текущего/кол-во_всех». Где-то по центру шаблона необходимо вставить Фамилию и Имя адресата.

Для сгенерированных файлов создайте отдельную папку в корне проекта.

Пример файла-шаблона:



```
message_template_1.txt
<почта>
Здравствуйте, <ФИ>!
Вы приглашены на мероприятие!
Мероприятие состоится в 18:00.
<номер>
```

Пример файла с именами:



personal_data.txt

```
BreakInvention@gmail.com Калинин Кирилл  
MemoryUniverse@yandex.ru Зимина Амина  
SealSidewalk@mail.ru Уткина Анна  
MineBrokenFire@gmail.com Андреев Никита
```

Примеры сгенерированного письма:

```
BreakInvention@gmail.com  
Здравствуйте, Калинин Кирилл!  
Вы приглашены на мероприятие!  
Мероприятие состоится в 18:00.  
1/10
```

SealSidewalk@mail.ru



Уважаемый(ая) Уткина Анна!

Благодарим Вас за плодотворное сотрудничество и вклад в развитие нашей компании.
Приглашаем Вас на новогодний вечер, который мы устраиваем в честь наших партнеров, сотрудников и друзей.

В преддверии Нового 2021 года мы соберемся вместе, чтобы приятно провести время в праздничной атмосфере и пожелать друг другу успехов и новых побед.

Пусть наступающий год принесет Вам только приятные перемены, каждый день будет успешным в личной жизни и плодотворным в работе!

Ждём вас 30 декабря в 104 аудитории в 17:00.

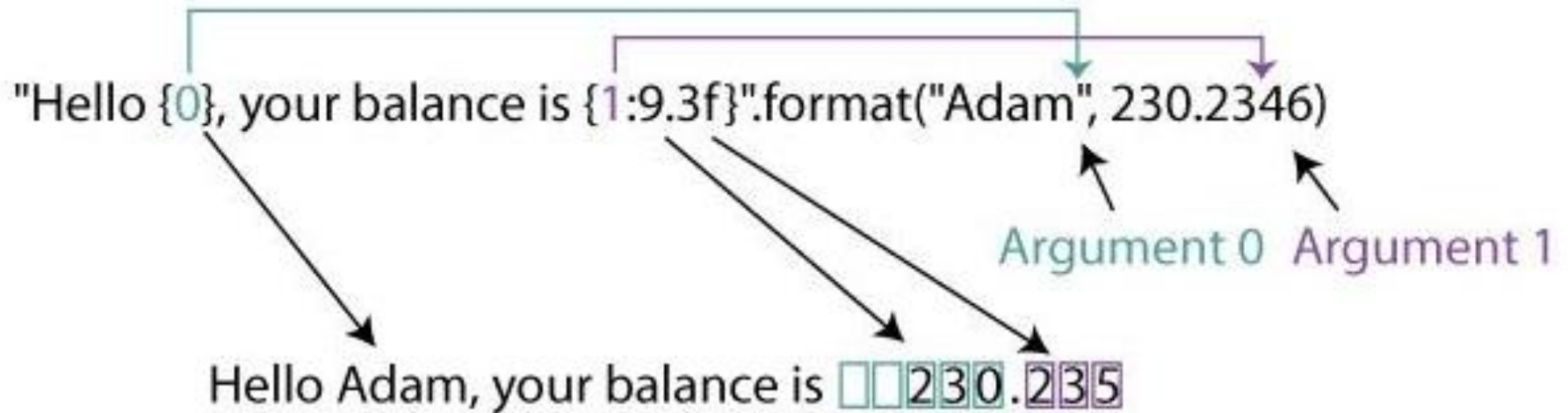
3/10



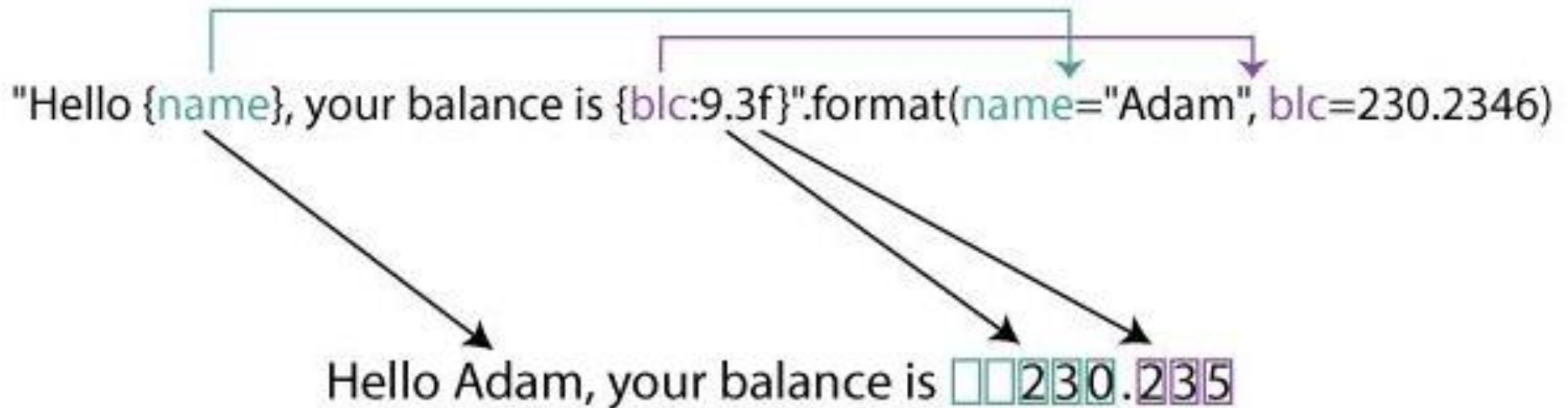
`.format()`



Для позиционных аргументов



Для аргументов-ключевых слов





Исключения try / except

Классы исключений и их обработка:

Ранее мы не раз сталкивались с исключениями в Python. Для начала давайте попробуем вызвать исключение и посмотрим, что при этом произойдет:

1/0

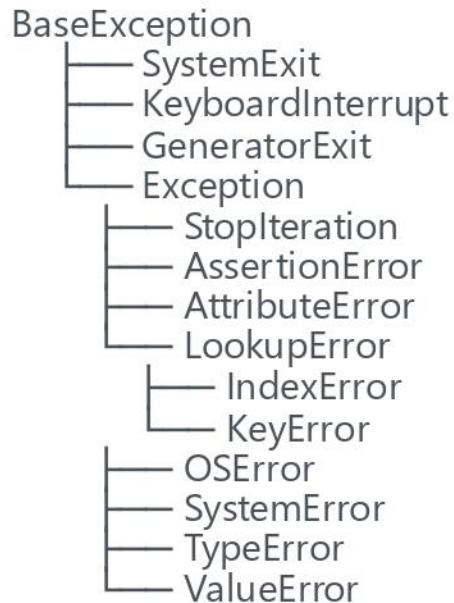
```
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero
```

При делении на 0 возникло исключение, и в этом случае на стандартный вывод печатается информация о его типе (в нашем случае это `ZeroDivisionError`), дополнительная информация об исключении, а также стек вызовов.

В этом примере стек вызовов очень небольшой. Когда исключение возникает в реальной программе, в стеке вызовов мы можем увидеть всю последовательность вызовов функций, которая привела к исключению.

Классы исключений и их обработка:

В Python есть два больших типа исключений. Первый — это исключения из стандартной библиотеки в Python, второй тип исключений — это пользовательские исключения. Они могут быть сгенерированы и обработаны самим программистом при написании программ на Python. Давайте посмотрим на иерархию исключений в стандартной библиотеке Python. Все исключения наследуются от базового класса `BaseException`:



Существуют несколько системных исключений, например, `SystemExit` (генерируется, если мы вызвали функцию `OSExit`), `KeyboardInterrupt` (генерируется, если мы нажали сочетание клавиш `Ctrl + C`) и так далее. Все остальные исключения генерируется от базового класса `Exception`. Именно от этого класса нужно породить свои исключения.

Классы исключений и их обработка:



Давайте посмотрим и обсудим некоторые исключения из стандартной библиотеки, такие как, например, `KeyError`, `IndexError`, `TypeError`, и попробуем их вызвать.

Ищем значение по несуществующему ключу в словаре, получаем `KeyError`:

```
d = {"foo": 1}
d["bar"]
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'bar'
```

Обратимся к несуществующему индексу в списке, получим `IndexError`:

```
l = [1,2]
l[10]
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Обратимся к несуществующему индексу в списке, получим `IndexError`:

```
1 + "10"
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Классы исключений и их обработка:

Если исключение сгенерировано, Python-интерпретатор остановит свою работу и на экран будет выведен стек вызовов и информация о типе исключений. Чтобы программа не останавливала работу, можно обработать исключение при помощи блока `try except`. То есть код, который потенциально может генерировать исключения, мы обрамляем в блок `try except`, и при генерации исключений управление будет передано в блок `except`. Таким образом можно отловить все исключения, которые генерируются в блоке `try except`:

```
try:  
    1 / 0  
except Exception:  
    print("Ошибка")
```

В блоке `except` можно указать тип исключения (в данном случае `Exception`), чтобы отлавливать исключения всех типов, у которых класс этого типа является родителем. В целом неправильно ждать любые исключения, и это может привести к непредвиденным сюрпризам работы вашей программы. Посмотрим на примере корректного `except`:

```
while True:  
    try:  
        raw = input("введите число: ")  
        number = int(raw)  
        break  
    except ValueError:  
        print("некорректное значение")
```

Классы исключений и их обработка:

Также у блока `try` `except` может быть блок `else`. Блок `else` вызывается в том случае, если никакого исключения не произошло:

```
while True:
    try:
        raw = input("введите число: ")
        number = int(raw)
    except ValueError:
        print("некорректное значение")
    else:
        break
```

Классы исключений и их обработка:

Если нам нужно обработать несколько исключений, мы можем использовать несколько блоков `except` и указать разные классы для обработки исключения. Причем в каждом блоке `except` может быть свой собственный обработчик:

```
while True:
    try:
        raw = input("введите число: ")
        number = int(raw)
        break
    except ValueError:
        print("некорректное значение!")
    except KeyboardInterrupt:
        print("выход") break
```


Классы исключений и их обработка:

Если обработчик исключений выглядит одинаково, то несколько исключений можно передать в виде списка в блок except:

```
total_count = 100_000
while True:
    try:
        raw = input("введите число: ")
        number = int(raw)
        total_count = total_count / number
        break
    except (ValueError, ZeroDivisionError):
        print("некорректное значение!")
```

Классы исключений и их обработка:



Бывает удобно пользоваться иерархией классов исключений. Посмотрим на два класса исключений `IndexError` и `KeyError` и их родителя `LookupError`:

```
issubclass(KeyError, LookupError)
issubclass(IndexError, LookupError)
```

True

True

Благодаря наследованию мы можем обрабатывать сразу обе ошибки в следующей программе, которая получает надписи на футболках из базы данных, где они отсортированы по цвету:

```
database = { "red": ["fox", "flower"], "green": ["peace", "M", "python"] }
try:
    color = input("введите цвет: ")
    number = input("введите номер по порядку: ")
    label = database[color][int(number)]
    print("вы выбрали:", label)
# except (IndexError, KeyError):
except LookupError:
    print("Объект не найден")
```

Классы исключений и их обработка:

Также у исключений есть дополнительный блок `finally`. Рассмотрим проблему. Например, мы открываем файл, читаем строки, обрабатываем эти строки, и в процессе работы нашей программы возникает исключение, которое мы не ждем. В таком случае файл закрыт не будет. Открытые файловые дескрипторы могут накапливаться, чего не следует допускать. Таким же образом могут накапливаться открытые сокеты или не освобождаться память. Для контроля таких ситуаций существуют, во-первых, контекстные менеджеры, а во-вторых, можно использовать блок `finally` в исключениях.

Мы пишем блок `finally` и вызываем метод `close()` для нашего объекта `f`. Возникло исключение или не возникло — блок `finally` будет выполнен и файл закроется:

```
f = open("list.txt")
try:
    for line in f:
        print(line.rstrip("\n"))
        1 / 0
except OSError:
    print("ошибка")
finally:
    f.close()
```


Генерация исключений(на будущее):



Для получения доступа к объекту исключений, нам необходимо воспользоваться конструкцией `except ... as err`. В следующем примере, если будет сгенерировано исключение `OSError`, то сам объект исключений будет связан с переменной `err` и эта переменная `err` будет доступна в блоке `except`. У каждого объекта типа исключений есть свои свойства, например, `errno` и `strerror` — это строковое описание ошибки и код ошибки. При помощи этих атрибутов можно получать доступ и обрабатывать исключения нужным вам образом.

```
try:
    with open("/file/not/found") as f:
        content = f.read()
except OSError as err:
    print(err.errno, err.strerror)
```

Генерация исключений(на будущее):



При вызове исключения можно передать ему любые аргументы, которые потом будут доступны как атрибут `args`. Предположим, продолжая предыдущий пример, что в момент генерации исключения `ValueError` мы передали туда строку и имя файла. Теперь в блоке `except` обратиться к атрибуту `args` объекта исключения — это и будет список наших параметров:

```
import os.path
filename = "/file/not/found"
try:
    if not os.path.exists(filename):
        raise ValueError("файл не существует", filename)
except ValueError as err:
    message, filename = err.args[0], err.args[1]
    print(message, code)
```

Иногда нам может потребоваться стек вызовов при обработке исключения. Стек вызовов можно получить при помощи модуля `traceback`, вызвав метод `print_exc`:

```
import traceback
try:
    with open("/file/not/found") as f:
        content = f.read()
except OSError as err:
    trace = traceback.print_exc()
    print(trace)
```

Генерация исключений(на будущее):



Как вы, возможно, уже догадались, исключение генерируется инструкцией `raise`. Для генерации исключения мы должны написать `raise` и указать класс исключения. Также можно указывать не только класс, но и объект исключения, указав ему дополнительные свойства. Как уже было сказано, к этим свойствам потом можно будет обратиться через объект исключения при помощи атрибута `args`. В следующем примере мы проверяем, что пользователь ввёл число, в противном случае генерируем исключение и затем в блоке `except` обрабатываем его:

```
try:
    raw = input("введите число: ")
    if not raw.isdigit():
        raise ValueError
except ValueError:
    print("некорректное значение!")
```

```
try:
    raw = input("введите число: ")
    if not raw.isdigit():
        raise ValueError("плохое число", raw)
except ValueError as err:
    print("некорректное значение!", err)
```

Генерация исключений(на будущее):



Иногда, поймав исключение, мы хотим делегировать обработку этого исключения другим функциям — тем, который вызвали данную функцию. Для этого нужно использовать инструкцию `raise` без параметров. В следующем примере в случае некорректного числа на экран выводится надпись: "Некорректное значение", и при помощи инструкции `raise` мы делегируем исключение на уровень выше. Если мы попробуем исполнить данный код, интерпретатор покажет на стандартный вывод информации об этом исключении и прекратит работу программы.

```
try:
    raw = input("введите число: ")
    if not raw.isdigit():
        raise ValueError("плохое число", raw)
except ValueError as err:
    print("некорректное значение!", err)
    # делегирование обработки исключения
    raise
```



Задачи



Задача 1

Найдите потенциальные источники ошибок.

Используя конструкцию **try** добавьте в код обработку соответствующих исключений.

```
def avg(a, b):  
    return (a * b) ** 0.5  
  
a = float(input("a = "))  
b = float(input("b = "))  
c = avg(a, b)  
print("Среднее геометрическое = {:.2f}".format(c))
```



Задача 13.

Дмитрий считает, что когда текст пишут в скобках (как вот тут, например), его читать не нужно.

Вот и надумал он существенно укоротить время чтения, написав функцию, которая будет удалять все, что расположено внутри скобок.

Помогите ленивому Диме разработать функцию `shortener(st)`, которая будет удалять все, что внутри скобок и сами эти скобки, возвращая очищенный текст (скобки могут быть вложенными).

```
print(shortener('Падал(лишнее (лишнее) лишнее) прошлогодний снег (лишнее)'))  
print(shortener('(лишнее(лишнее))Падал прошлогодний (лишнее(лишнее(лишнее)))снег'))
```



Домашнее задание



Задание 1.

Александр решил каким-то образом отобразить в тексте BACKSPACE (т.е. удаление последнего символа).

Он подумал, что символ «@» отлично для этого подходит.

Напишите функцию `cleaned_str(st)`, которая будет выполнять BACKSPACE в его строках:

`гр@оо@лк@оц@ва` -> голова

`сварка@@@@@лоб@ну@` -> слон

```
print(cleaned_str('гр@оо@лк@оц@ва'))  
print(cleaned_str('сварка@@@@@лоб@ну@'))
```



Входит в ГК Аплана



АКАДЕМИЯ АЙТИ

Основана в 1995 г.

E-learning
и очное
обучение

Направления обучения:

Информационные технологии

Информационная безопасность

ИТ-менеджмент и управление проектами

Разработка и тестирование ПО

Гос. и муниципальное управление

Филиалы:

Санкт-Петербург, Казань, Уфа, Челябинск,
Хабаровск, Красноярск, Тюмень, Нижний
Новгород, Краснодар, Волгоград, Ростов-на-Дону



Ежегодные награды
Microsoft,
Huawei, Cisco и
другие

Головной офис
в Москве

Разработка
программного
обеспечения и
информационных
систем

Программы по
импортозамещению

Ресурсы более 400
высококласных
экспертов и
преподавателей

Сеть региональных учебных центров
по всей России

Крупные заказчики



100+

сотрудников



АКАДЕМИЯ АЙТИ



Спасибо за внимание!

Центральный офис:

Москва, Варшавское шоссе 47, корп. 4, 7 этаж

Тел: +7 (495) 150-96-00

academy@it.ru

academyit.ru