Ввод-вывод. Работа с файлами.

Исключения

```
import traceback

def function_with_exception():
    # raise Exception() # плохо, не делайте так
    # raise "informative text" # плохо, не делайте так
    raise ValueError("informative text") # хорошо

# можно поймать это исключение:
try:
    function_with_exception()
except ValueError as e:
    traceback.print_tb(e.__traceback__)
finally:
    print("finally")
```

Почитать об основных исключениях можно здесь

Ввод-вывод

`input` — считать строку из стандартного потока ввода. Символы новой строки будут обрезаны.

`print` — (по-умолчанию) вывести значение в стандартный поток вывода

```
def hello_world():
    hello = input("Say hello to the world: ")
    return f"{hello} world"


print('<-','-' * 5, hello_world(), '-' * 5, '->',  sep='|',
end='\n:c\n') #?
```

Перенаправление потоков

Интерпретатор, исполняющий скрипт — это полноценный процесс со своим набором файловых дескрипторов.

Исполним код с предыдущего слайда:

```
$ echo 'Goodbye' > goodbye_file
$ python script.py < goodbye_file
Say hello to the world: <-|-----|Goodbye world|-----|->
:c

$ echo 'Goodbye' | python script.py
Say hello to the world: <-|-----|Goodbye world|-----|->
:c
```

Работа с файлами.

Файл представлен специальным объектом (как и все в python).

Открыть файл можно с помощью `open`. Первый аргумент — путь к файлу, второй (опциональный) — режим, в котором будет открыт файл:

- 'r' — read (default),
- 'w' — write,
- 'a' — append,
- 'r+' — read and write,
- 'a+' — read and append;

Работа с файлами.

И есть два дополнительных режима:

- 'b' — binary mode,
- 't' — text mode (default)

первый нужно использовать, если нужно работать с файлом как с бинарными данными и ничего не декодировать.

Второй, если нужно декодировать и работать с текстом.

Файлы, которые вернет `open` будут оперировать с байтовыми объектами или строками соответственно.

Работа с файлами

```
f.read() # считать весь файл
f.read(size) # считать не более size символов/байт

f.readline() # считать строку из файла

f.write('This is a test\n') # записать строку в файл


# для работы с бинарными файлами
f.tell() # текущая позиция
f.seek(5) # перейти на 5 байт вперед от текущей позиции
```

Работа с файлами.

После работы с файлом его обязательно нужно закрывать!

```
f = open('file', 'wt')
print(type(f))
# _io.TextIOWrapper

f.write('Ahahaha\n')
f.close()

f = open('file', 'r')
print(f.read())
# Ahahaha
f.close()
```

Работа с файлами. with

with-блок позволяет автоматически открывать/закрывать ресурсы (и не только) перед/после выполнения кода, в том числе закрывать используемые файлы.

Настоятельно рекомендуется использовать `with` всегда при работе с файлами.

```
with open('file', r) as file:
    # по файлу тоже можно итерироваться
    for line in file:
        print(line)
# не нужно закрывать файл, with сам его закроет
```

Дескрипторы

`fdopen` — создает файловый объект, связанный с конкретным файловым дескриптором. Алиас на `open`, но принимает номер дескриптора в качестве первого аргумента

```
import os

def print_from_descriptor(fd):
    with os.fdopen(fd) as file:
        for line in file:
            print(line, end='')

print_from_descriptor(4)
```

Дескрипторы

запустим скрипт с предыдущего слайда:

```
$ printf "1\n2\n\3n" > file
$ python script.sh 4<file
1
2
3
```

Дескрипторы

Для стандартных потоков файловые объекты уже есть, в модуле sys:

- sys.stdin,
- sys.stdout,
- sys.stderr.

`input` и `print` используют `sys.stdin` и `sys.stdout`.

Все ошибки интерпретатор пишет в `sys.stderr`.

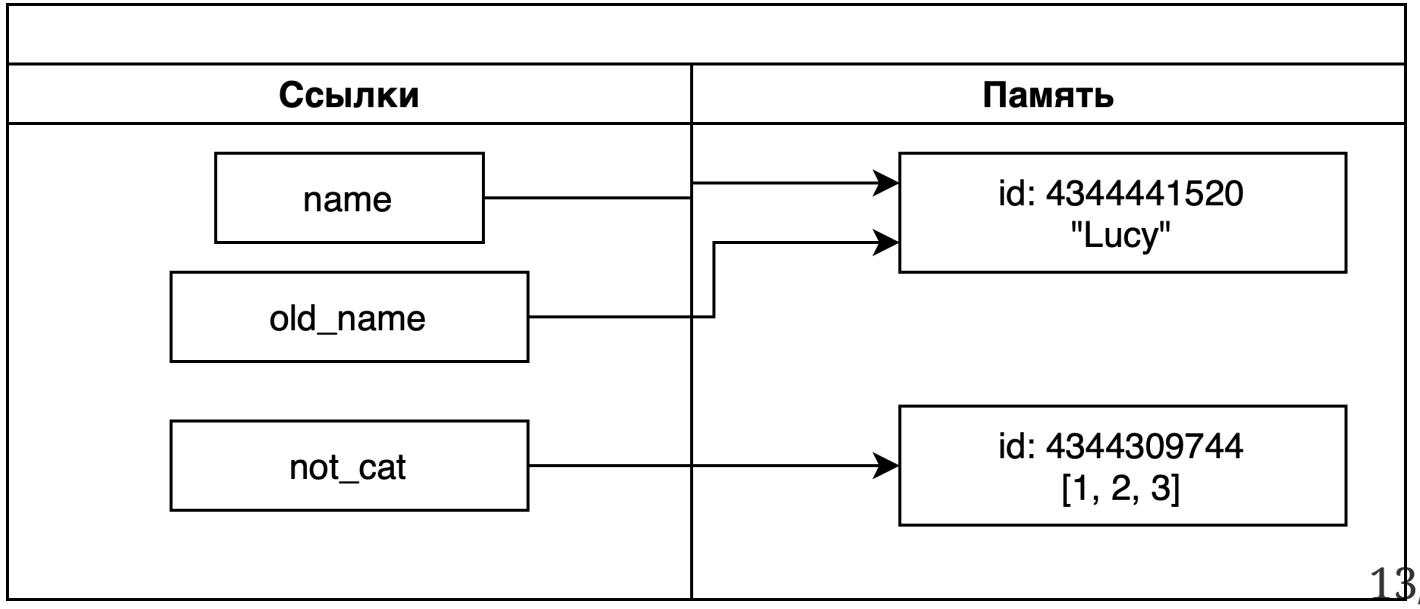
Дескрипторы

Будьте осторожны и не используйте `with` со стандартными потоками:

```
import sys

with sys.stdin as file:
     file.read()

print(input())
# Traceback (most recent call last):
#     print(input()) #?
# ValueError: I/O operation on closed file.
```

Про ссылки

```
name = "Lucy" # создали объект строку и связали его со ссылкой name
old_name = name # скопировали ссылку на тот же объект

not_cat = [1, 2, 3] # создали новый объект и связали со ссылкой not_cat
```



Про ссылки

Если объект изменяемый, то изменить его можно по любой ссылке:

```
name_letters = list("Lucy")

old_name = name_letters
old_name[2] = 10

print(name_letters) #?
```

Про ссылки

Если список создать с помощью `*`, то **в нем окажутся копии ссылок**, которые лежали в исходном списке, **не копии значений**

Если объект хотя бы по одной ссылке окажется изменяемым, то могут возникнуть проблемы:

```
a = [3] * 3
a[2] = 1
print(a) #?

a = [[3] * 3] * 3
a[2, 1] = 1
print(a) #?
```