



MACHINE LEARNING

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON И
РАЗРАБОТКА ПРОГРАММ ДЛЯ МАШИННОГО ОБУЧЕНИЯ
ЛЕКЦИЯ VI

План занятия

- Файловая система и работа с файлами
- Домашние задания: как реализуются программы

Открытие и закрытие файла в Python

Open()

При вызове, эта функция создает объект типа файл, с которым в дальнейшем можно работать.

Синтаксис функции `open()` в Python.

```
my_file = open(имя_файла [, режим_доступа][, буферизация])
```

При этом:

- `имя_файла`: строка, содержащая имя файла с расширением. Например, "my_file.txt".
- `режим_доступа`: строка, которой мы указываем для чего открывается файл: для чтения, записи, добавления информации, и т.д. Например, "w". По умолчанию файл открывается для чтения - "r".
- `буферизация`: Целое число. Если значение аргумента указано 0 - файл открывается без буферизации, 1 с построчной буферизацией, больше одного процесс буферизации выполняется с указанным размером буфера. Отрицательное число - размер буфера будет равен системному.

Список режимов доступа к файлу в Python

r	Открывает файл только для чтения. Указатель стоит в начале файла.
rb	Открывает файл для чтения в двоичном формате. Указатель стоит в начале файла.
r+	Открывает файл для чтения и записи. Указатель стоит в начале файла.
rb+	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла.
w	Открывает файл только для записи. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
wb	Открывает файл для записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
w+	Открывает файл для чтения и записи. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.
wb+	Открывает файл для чтения и записи в двоичном формате. Указатель стоит в начале файла. Создает файл с именем имя_файла, если такового не существует.

a	Открывает файл для добавления информации в файл. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
ab	Открывает файл для добавления в двоичном формате. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
a+	Открывает файл для добавления и чтения. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.
ab+	Открывает файл для добавления и чтения в двоичном формате. Указатель стоит в конце файла. Создает файл с именем имя_файла, если такового не существует.

Атрибуты файлового объекта в Python

Как только файл был открыт и у вас появился файловый объект, вы можете получить следующую информацию о нем:

file.closed	Возвращает True если файл был закрыт.
file.mode	Возвращает режим доступа, с которым был открыт файл.
file.name	Возвращает имя файла.
file.softspace	Возвращает False если при выводе содержимого файла следует отдельно добавлять пробел.

Например:

```
my_file = open("some.txt", "w")
print("Имя файла: ", my_file.name)
print("Файл закрыт: ", my_file.closed)
print("В каком режиме файл открыт: ", my_file.mode)
print("Пробелы: ", my_file.softspace)
```

Заккрытие файла

Метод close()

Метод файлового объекта `close()` автоматически закрывает файл, при этом теряется любая несохраненная информация. Работать с файлом (читать, записывать) после этого нельзя.

Python автоматически закрывает файл если файловый объект к которому он привязан присваивается другому файлу. Однако, хорошей практикой будет вручную закрывать файл командой `close()`.

```
my_file = open("some.txt")
print("Имя файла: ", my_file.name)
print("Файл закрыт: ", my_file.closed)
my_file.close()
print("А теперь закрыт: ", my_file.closed)
```

Чтение и запись файлов

Запись файлов. Метод `write()`

Метод `write()` записывает любую строку в открытый файл. Важно помнить, что строки в Python могут содержать двоичные данные, а не только текст.

Метод `write()` не добавляет символ переноса строки (`'\n'`) в конец файла.

Синтаксис метода `write()`:

`my_file.write(string);`

Например:

```
my_file = open("some.txt", "w")
my_file.write("Мне нравится Python!\nЭто классный язык!")
my_file.close()
```

Вышеприведенный код создаст файл `some.txt` и запишет в него указанную строку.

Чтение и запись файлов

Чтение из файла в Python. Метод `read()`.

Метод `read()` читает строку из открытого файла.

Синтаксис метода `read()`:

```
my_file.read([count])
```

Необязательный параметр `count` - это количество байт, которые следует прочитать из открытого файла. Этот метод читает информацию с начала файла и, если параметр `count` не указан, до конца файла.

Например, прочтем созданный нами файл
some.txt:

```
my_file = open("some.txt")
my_string = my_file.read()
print("Было прочитано:")
print(my_string)
my_file.close()
```

```
>>>
Было прочитано:
Мне нравится Python!
Это классный язык!
>>> |
```

Метод tell()

После того как вы вызвали метод `read()` на файловом объекте, если вы повторно вызовете `read()`, то увидите лишь пустую строку. Это происходит потому, что после первого прочтения указатель находится в конце файла. Для того чтобы узнать позицию указателя можно использовать метод `tell()`.

```
my_file = open("some.txt")
my_file.read(10)
print ("Я на позиции:", my_file.tell())
my_file.close()
```

Метод `tell()` сообщает в скольких байтах от начала файла мы сейчас находимся

Метод seek()

Чтобы перейти на нужную нам позицию, следует использовать метод `seek()`.

Синтаксис метода `seek()`: `my_file.seek(offset, [from])`

Аргумент `offset` указывает на сколько байт перейти. опциональный аргумент `from` означает позицию, с которой начинается движение. 0 - означает начало файла, 1 - нынешняя позиция, 2 - конец файла.

Метод `seek (offset [, from])` изменяет текущую позицию файла. Аргумент смещения указывает количество перемещаемых байтов. Аргумент `from` указывает ссылочную позицию, из которой должны быть перемещены байты.

Например:

```
my_file = open("some.txt", "r")
print(my_file.read(10))
print("Мы находимся на позиции: ", my_file.tell())
# Возвращаемся в начало
my_file.seek(0)
print(my_file.read(10))
my_file.close()
```

Метод write()

Если вы хотите не перезаписать файл полностью (что делает метод `write` в случае открытия файла в режиме `'w'`), а только добавить какой-либо текст, то файл следует открывать в режиме `'a'` - `appending`. После чего использовать все тот же метод `write`.

Например

:

```
# Удалит существующую информацию в some.txt и запишет "Hello".  
my_file = open("some.txt", 'w')  
my_file.write("Hello")  
my_file.close()  
# Оставит существующую информацию в some.txt и добавит "Hello".  
my_file = open("some.txt", 'a')  
my_file.write("Hello")  
my_file.close()
```

Блок try-except В Python

Уязвимый код заключается в **блок try**, после которого следует **блок except**, которому может задаваться возможная ошибка и реакция на нее:

```
try:  
    a = float(input("Введите число:"))  
except ValueError:  
    print ("Это не число!")
```

В данном примере программа пытается конвертировать информацию введенную пользователем в тип float, если же при этом возникнет ошибка класса ValueError, то выводится строка "This is not a valid number". В блоке except мы можем задать те классы ошибок на которые данный блок должен сработать, если мы не укажем ожидаемый класс ошибок, то блок будет реагировать на любую возникшую ошибку.

Блок `try` может содержать неограниченное количество блоков `except`:

```
try:
    a = float(input("Введите число: "))
    print (100 / a)
except ValueError:
    print ("Это не число")
except ZeroDivisionError:
    print ("На ноль делить нельзя")
```


Кроме того мы можем добавить пустой блок **except**, который будет срабатывать на непредвиденную выше ошибку. Пустой блок **except** всегда должен идти последним:

```
try:
    a = float(input("Введите число: "))
    print(100 / a)
except ValueError:
    print("Это не число!")
except ZeroDivisionError:
    print("На ноль делить нельзя!")
except:
    print("Неожиданная ошибка.")
```

Блок else в блоке try-except в Python

Блоку except можно добавить необязательный блок else, который сработает в случае, если программа выполнена без ошибок:

```
try:
    a = float(input("Введите число: "))
    print (100 / a)
except ValueError:
    print ("Это не число!")
except ZeroDivisionError:
    print ("На ноль делить нельзя!")
except:
    print ("Неожиданная ошибка.")
else:
    print ("Код выполнен без ошибок")
```

Блок finally в Python

Также у блока `except` есть еще один необязательный блок **finally**, который сработает независимо от того, выполнился код с ошибками или без:

```
try:
    a = float(input("Введите число: "))
    print(100 / a)
except ValueError:
    print("Это не число!")
except ZeroDivisionError:
    print("На ноль делить нельзя!")
except:
    print("Неожиданная ошибка.")
else:
    print("Код выполнен без ошибок")
finally:
    print("Я выполняюсь в любом случае!")
```

The background is a blue gradient with decorative white circuit-like lines in the corners. The lines consist of straight segments and small circles, resembling a printed circuit board layout.

Спасибо за внимание!