Python akademie - lekce 9 - 12.12.2024

09_01: 🗒 Textové soubory

Zajímavé odkazy z této lekce:

- Oficiální dokumentace k **docstring** u funkci (python.org)
- Oficiální dokumentace k **UTF-8** u funkci (wikipedia.org)

Pojem file io

Doposud jsme pracovali pouze s objekty Pythonu vlastními nebo s knihovnami. Dneska si povíme něco o práci **s textovými soubory** přímo u vás na počítači.

Celkově pokud budete v rámci Pythonu pracovat se soubory (obecně), mluvíme o procesu `io` (někdy také `i/o`).

Toto označení vychází z anglického *input* a *output*. Tedy **vstup** a **výstup**.

V průběhu této lekce se naučíme textové soubory **číst**, **zapisovat** i **upravovat** tak, aby to za nás vždy provedl interpret Pythonu.

Textové soubory (~text files)

Textovým souborem rozumějme jakýkoliv soubor, který má příponu `.txt`.

Jeden takový textový soubor máme k dispozici v aktuálním adresáři:

Práce s textovými soubory nám může být v principu jasná (v pracovním prostředí počítače).

Nyní se ale pojďme naučit pracovat s textovým editorem pomocí Pythonu (pomocí interpretu Pythonu).

Práce s textovými soubory (~File I/O)

Chceme napsat šikovnější programy, které umí Pythonu vysvětlit, aby pracoval s různymi textovými soubory.

Nejprve ale budeme muset Python přesvědčit, aby nám soubor **vytvořil**.

Vytvoření nového souboru (zápis)

```
muj_string = "Python je cool!"
```

String `muj_string` máme aktuálně k dispozici pouze jako nějaký **objekt Pythonu** (v aktuálním prostředí jako `str`).

Jak jej ale uložit do skutečného textového souboru na našem disku?

Nejprve potřebujeme "zevnitř" Pythonu (~interpreta Pythonu), vytvořit soubor. Nejdříve ověříme zabudované funkce.

```
help(open)
```

```
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closef
    Open file and return a stream.  Raise OSError upon failure.

    file is either a text or byte string giving the name (and the path
    if the file isn't in the current working directory) of the file to
    be opened or an integer file descriptor of the file to be
    wrapped. (If a file descriptor is given, it is closed when the
    returned I/O object is closed, unless closefd is set to False.)

    mode is an optional string that specifies the mode in which the file
    is opened. It defaults to 'r' which means open for reading in text
    mode.  Other common values are 'w' for writing (truncating the file if
    it already exists), 'x' for creating and writing to a new file, and
    'a' for appending (which on some Unix systems, means that all writes
    append to the end of the file regardless of the current seek position).
    In text mode, if encoding is not specified the encoding used is platform
    dependent: locale.getpreferredencoding(False) is called to get the
    current locale encoding. (For reading and writing raw bytes use binary
    mode and leave encoding unspecified.) The available modes are:

    ========= ========================================================
    Character Meaning
    --------- --------------------------------------------------------
    'r'       open for reading (default)
    'w'       open for writing, truncating the file first
    'x'       create a new file and open it for writing
    'a'       open for writing, appending to the end of the file if it exists
    'b'       binary mode
    't'       text mode (default)
    '+'       open a disk file for updating (reading and writing)
    'U'       universal newline mode (deprecated)
    ========= ========================================================

    The default mode is 'rt' (open for reading text). For binary random
    access, the mode 'w+b' opens and truncates the file to 0 bytes, while
    'r+b' opens the file without truncation. The 'x' mode implies 'w' and
    raises an `FileExistsError` if the file already exists.

    Python distinguishes between files opened in binary and text modes,
    even when the underlying operating system doesn't. Files opened in
    binary mode (appending 'b' to the mode argument) return contents as
    bytes objects without any decoding. In text mode (the default, or when
    't' is appended to the mode argument), the contents of the file are
    returned as strings, the bytes having been first decoded using a
    platform-dependent encoding or using the specified encoding if given.

    'U' mode is deprecated and will raise an exception in future versions
    of Python.  It has no effect in Python 3.  Use newline to control
    universal newlines mode.

    buffering is an optional integer used to set the buffering policy.
    Pass 0 to switch buffering off (only allowed in binary mode), 1 to select
    line buffering (only usable in text mode), and an integer > 1 to indicate
    the size of a fixed-size chunk buffer.  When no buffering argument is
    given, the default buffering policy works as follows:
```

```
muj_soubor = open('novy_soubor.txt', mode='w')
```

Soubor si můžeme otevřít, ale zjistíme, že je v tento moment prázdný.

Funkce `open` pouze **vytvoří** (~iniciuje) nový objekt `muj_soubor`. Hodnotou tohoto souboru je zapsaní **skutečného souboru** (jehož jméno jsme uvedli v závorce s příslušným argumentem) na váš disk, do aktuálního otevřeného adresáře.

Příslušný text teprve musíme zapsat.

```
muj_soubor.write(muj_string)
```

```
15
```

Číslo `15`, které vidíme na výstupu je počet zapsaných znaků (*~bytes*). (zdroj)

Pojďme si nyní společně **prohlédnout zapsaný soubor** na disku.

Protože jsme do souboru zapisovali, ale **neukončili jej**, nejsme schopni s ním ještě manipulovat. Pomocí metody `closed` ověříme, jestli je spojení ukončené. (zdroj)

```
muj_soubor.closed
```

```
False
```

Pokud zjistíme, že není ukončené, ukončíme jej pomocí metody `close`. (zdroj)

```
muj_soubor.close()
```

```
muj_soubor.closed
```

```
True
```

Teprve po ukončení *streamu* objektu můžeme soubor `novy.txt` prozkoumat.

Nyní chceme doplnit **další řádek** v našem souboru:

```
muj_string2 = "Prave probirame lekci 9"

muj_soubor = open("novy_soubor.txt", mode="w")

muj_soubor.write(muj_string2)
```

```
23
```

```
muj_soubor.close()
```

Zkontrolujeme jak vypadá nově přidaný string.

Opatrně na `mode="w"` . Pokud opětovně načtete stejný soubor v tomto režimu, přesunute "zapisovač" (představ si jej jako blikající kurzor v editoru) opět na začátek souboru.

Interpret ale zapisuje od místa, kde se zapisovač nachází, takže dojde k **přepsání stávajícího obsahu**.

Pokud chceš automaticky zapisovat nehledě na umístění našeho *zapisovač*, otevři soubor s argumentem `mode="a"` , tedy v režimu **append**. (zdroj)

```
muj_string = "Python je cool!"
muj_string2 = "Prave probirame lekci 9"

muj_soubor = open("novy_soubor.txt", mode="w")

muj_soubor.write(muj_string)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10528/4086508546.py in <module>
----> 1 muj_soubor.write(muj_string, muj_string2)

TypeError: TextIOWrapper.write() takes exactly one argument (2 given)
```

```
muj_soubor.close()
```

```
muj_soubor = open("novy_soubor.txt", mode="a")  # rezim 'append'
```

```
muj_soubor.write(muj_string2)
```

```
23
```

```
muj_soubor.close()
```

Přidáním `\n` na konec stringů dosáhneme toho, že následující string bude vypsaný **na novém řádku**.

```
!rm novy_soubor.txt  # pouze příkaz do linuxové přík. řádky
```

```
!del novy_soubor.txt  # ve Windows
```

```
muj_str_1 = "Ahoj, ja jsem Matous\n"
muj_str_2 = "Rad ctu, hraji na klavir\n"
muj_str_3 = "A co ty?:)"
```

```
muj_soubor = open("novy_soubor.txt", mode="w")
```

```
muj_soubor.write(muj_str_1)
muj_soubor.write(muj_str_2)
muj_soubor.write(muj_str_3)
```

```
10
```

```
muj_soubor.close()
```

```
muj_soubor2 = open('druhy_soubor.txt', mode='w')
```

```
muj_soubor2.close()
```

Čtení existujícího souboru

Zatím jsme tu nahlíželi na obsah našich textových souborů pouze pomocí nějakého grafického prohlížeče. Tentokrát si pojďme zkusit čtení pomocí **Pythonu**.

Opět použijeme zabud. funkci `open` .

```
muj_existujici_soubor = open("novy_soubor.txt")
```

```
print(muj_existujici_soubor)
```

```
<_io.TextIOWrapper name='novy_soubor.txt' mode='r' encoding='cp1252'>
```

```
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)
```

```
obsah_txt = muj_existujici_soubor.read()
```

```
print(obsah_txt)
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)
```

Opět, pokud jedenkrát přečtete obsah celého souboru, *zapisovač* (*~kurzor*) přečte postupně celý text a zůstane na konci souboru.

Proto při dalším čtení získáme prázdný výstup, protože kurzor neprojde text znovu.

```
muj_existujici_soubor.seek(0)     # přesune kurzor na začátek souboru
```

```
0
```

```
muj_existujici_soubor.seek(0, 2)  # přesune kurzor na konec souboru
```

```
58
```

```
muj_existujici_soubor.seek(0)     # přesune kurzor na začátek souboru
```

```
0
```

```
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)
```

```
muj_existujici_soubor.seek(0)
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)
```

Reprezentace znaků

```
ord("\n")  # vrací ze znaku číselné označení (Unicode standart)
```

```
10
```

```
chr(10)   # vrací z číselného označení znak
```

```
'\n'
```

Metody pro čtění obsahu `TextIOWrapper` objektu:

1. `read` - přečte celý soubor jako jeden string
2. `readline` - přečte pouze první řádek jako string
3. `readlines` - přečte celý soubor jako list (co řádek, to údaj)

Ukázka různých variant

```
muj_existujici_soubor.seek(0)
print(muj_existujici_soubor.readline())
```

```
Ahoj, ja jsem Matous
```

```
print(muj_existujici_soubor.readline())
```

```
Rad ctu, hraji na klavir
```

```
print(muj_existujici_soubor.readline())
```

```
A co ty?:)
```

```
print(muj_existujici_soubor.readline())
```

```
muj_existujici_soubor.seek(0)
```

```
0
```

```
print(muj_existujici_soubor.readlines())
```

```
['Ahoj, ja jsem Matous\n', 'Rad ctu, hraji na klavir\n', 'A co ty?:)']
```

```
muj_existujici_soubor.close()
```

Současně zapisovat a číst

Vhodnou hodnotou argumentu `mode` můžeme specifikovat režim, kdy můžeme jak zapisovat, tak číst:

```
muj_existujici_soubor.closed
```

```
True
```

```
muj_existujici_soubor = open("novy_soubor.txt", mode="r+")
```

```
help(open)
```

```
Help on built-in function open in module io:

open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closef
    Open file and return a stream.  Raise OSError upon failure.

    file is either a text or byte string giving the name (and the path
    if the file isn't in the current working directory) of the file to
    be opened or an integer file descriptor of the file to be
    wrapped. (If a file descriptor is given, it is closed when the
    returned I/O object is closed, unless closefd is set to False.)

    mode is an optional string that specifies the mode in which the file
    is opened. It defaults to 'r' which means open for reading in text
    mode.  Other common values are 'w' for writing (truncating the file if
    it already exists), 'x' for creating and writing to a new file, and
    'a' for appending (which on some Unix systems, means that all writes
    append to the end of the file regardless of the current seek position).
    In text mode, if encoding is not specified the encoding used is platform
    dependent: locale.getpreferredencoding(False) is called to get the
    current locale encoding. (For reading and writing raw bytes use binary
    mode and leave encoding unspecified.) The available modes are:

    ========= ================================================================
    Character Meaning
    --------- ----------------------------------------------------------------
    'r'       open for reading (default)
    'w'       open for writing, truncating the file first
```

```
        'x'       create a new file and open it for writing
        'a'       open for writing, appending to the end of the file if it exists
        'b'       binary mode
        't'       text mode (default)
        '+'       open a disk file for updating (reading and writing)
        'U'       universal newline mode (deprecated)
        ========= ============================================================

        The default mode is 'rt' (open for reading text). For binary random
        access, the mode 'w+b' opens and truncates the file to 0 bytes, while
        'r+b' opens the file without truncation. The 'x' mode implies 'w' and
        raises an `FileExistsError` if the file already exists.

        Python distinguishes between files opened in binary and text modes,
        even when the underlying operating system doesn't. Files opened in
        binary mode (appending 'b' to the mode argument) return contents as
        bytes objects without any decoding. In text mode (the default, or when
        't' is appended to the mode argument), the contents of the file are
        returned as strings, the bytes having been first decoded using a
        platform-dependent encoding or using the specified encoding if given.

        'U' mode is deprecated and will raise an exception in future versions
        of Python.  It has no effect in Python 3.  Use newline to control
        universal newlines mode.

        buffering is an optional integer used to set the buffering policy.
        Pass 0 to switch buffering off (only allowed in binary mode), 1 to select
        line buffering (only usable in text mode), and an integer > 1 to indicate
        the size of a fixed-size chunk buffer.  When no buffering argument is
        given, the default buffering policy works as follows:
```

```
print(muj_existujici_soubor)
```

```
<_io.TextIOWrapper name='novy_soubor.txt' mode='r+' encoding='cp1252'>
```

```
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)
```

```
muj_existujici_soubor.write("A jeste jeden radek!")
```

```
20
```

```
print(muj_existujici_soubor.tell())
```

```
78
```

```
muj_existujici_soubor.seek(0)
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)A jeste jeden radek!
```

```
muj_existujici_soubor.write("\nPosledni radek!")
```

```
16
```

```
muj_existujici_soubor.seek(0)
print(muj_existujici_soubor.read())
```

```
Ahoj, ja jsem Matous
Rad ctu, hraji na klavir
A co ty?:)A jeste jeden radek!
Posledni radek!
```

```
muj_existujici_soubor.close()
```

Kontextový manažer

Pokud vám není příjemné myslet na neustálou proceduru otevřít soubor, provést s ním potřebnou práci a zavřít jej, je tu i jiná možnost.

Syntaxe s `with`

Pokud nechcete hlídat zavírání jednotlivých souborů, můžeme použít syntaxi s `with` (tedy kontextový manažer).(zdroj)

Jeho použití, jak si ukážeme nesouvisí pouze s textovými soubory, ale pří práci s nimi je to skvělý pomocník.

Jeho použití spočívá v tom, že pomocí různých volatelných objektů v Pythonu (jako jsou funkce, př. `open`) spustí konkrétní proces.

Ihned potom umí automaticky ukončit celý proces, pokud interpret nenajde žádné další odsazené ohlášení.

```
with open("novy_soubor.txt", mode="a") as muj_existujici_soubor:
    muj_existujici_soubor.write("\nPosledni radek z kont. manazera!")
```

```
muj_existujici_soubor.closed
```

```
True
```