Лабораторна робота №6
Модулі, методи, методи класу str
Короткі теоретичні відомості

Стандартна бібліотека Python містить 209 модулів та пакетів модулів. Модулі стандартної бібліотеки дозволяють вирішувати велику кількість різноманітних завдань. Якщо потрібно вирішити якусь завдання то програміст повинен в першу чергу звернути свою увагу на функціональні можливості модулів стандартної бібліотеки.

Коротко розглянемо модулі стандартної бібліотеки, які мають різне призначення.

Модуль `string` містить константи, які використовуються для роботи з типом даних str (рядок). Прикладом цих констант можуть бути `string.ascii_letters` та `string.hexdigits`.

Модуль `sys` надає доступ до об'єктів, з якими працює та які використовує інтерпретатор. Модуль також містить функції для взаємодії з інтерпретатором. Серед цих об`єктів потрібно звернути увагу на список `sys.argv`, який містить аргументи командного рядка. Якщо програма запущена на виконання командою з командного рядка то у списку буде як мінімум один елемент – рядок з назвою модуля.

Наприклад, якщо з командного рядка було запущено команду "`python demo.py one two three`", то можна отримати такий вивід:

```
>>> import sys
>>> print sys.argv
['demo.py', 'one', 'two', 'three']
```

Модуль `sys` має також атрибути `stdin, stdout` та stderr ("стандартний ввід", "стандартний вивід" та "стандартний вивід помилок" відповідно). Останній корисний для організації виведення попереджень і помилок при переспрямуванні `stdout`:

```
>>> sys.stderr.write('Попередження: файл для запису не знайдено; створюється новий файл')
Попередження: файл для запису не знайдено; створюється новий файл
```

Найпростіший шлях виходу з програми — це виклик функції `sys.exit()`.

Змінна `sys.path` — це список рядків, що визначають шляхи, які використовує інтерпретатор при пошуку модулів. Початкове значення цього списку - стандартне, що береться зі змінної середовища `PYTHONPATH` або із вбудованого стандартного значення (якщо `PYTHONPATH` не задано). Список `sys.path` можна змінювати за допомогою стандартних операцій зі списками.

Спробуйте самостійно дізнатися що міститься у наступних об`єктах модуля `sys` `sys.float_info`, `sys.int_info`, `sys.maxsize`, `sys.platform`, `sys.version`, `sys.getsizeof()`, `sys.getdefaultencoding()`, `sys.getrecursionlimit()`, `sys.modules`

Модуль `os` містить функції для отримання інформації і у деяких випадках для маніпуляції, локальними каталогами, файлами, процесами та змінними середовища. Модуль `os` підтримує інтерфейс з операційною системою. Наприклад, модуль `os` містить наступні функції взаємодії з операційною системою:

```
>>> import os
>>> os.system('time 0:02')
0
>>> os.getcwd() # Повертає поточну робочу директорію
'C:\\Python36'
>>> os.chdir('/server/accesslogs')
```

У випадку використання модуля os категорично не можна використовувати конструкцію "`from os import *`". Оскільки такі дії зумовлять перекривання вбудованої функції `open()` функцією `os.open()`, яка має зовсім інше призначення.

```
>>> dir(os) # повертає список усіх функцій модуля
```

Модуль `os` забезпечує незалежний від платформи доступ до засобів операційної системи. Змінна `os.environ` містить об`єкт, елементами якого є змінні середовища і їх значення. Робочий каталог програми можна отримати з допомогою функції `os.getcwd()`, а змінити його можна з допомогою функції `os.chdir()`. Також модуль містить функції для низькорівневої роботи з файлами на основі їх дескрипторів. Функцію `os.access()` можна використовувати для визначення наявності файлу та його доступності для читання або запису. Функція `os.listdir()` повертає список записів (імен файлів та каталогів, за винятком елементів . та ..) у вказаному каталозі. Функція `os.stat()` повертає різноманітні відомості про файл або каталог - режим доступу, час останнього звертання та розмір. Каталоги можуть створюватися за допомогою функції `os.mkdir()` або, якщо потрібно разом з проміжними каталогами, за допомогою функції `os.makedirs()`. Пусті каталоги можна видаляти за допомогою функції `os.rmdir()`, а дерева каталогів, які містять тільки пусті каталоги, - за допомогою функції `os.removedirs()`. Файли і каталоги можуть видалятися за допомогою функції `os.remove()`, а перейменовуватися за допомогою функції `os.rename()`. Функція `os.walk()` дозволяє виконати ітерації по всьому дереву каталогів, почергово добуваючи імена всіх файлів і каталогів. Окрім цих функцій, модуль `os` містить багато низькорівневих залежних від платформи функцій. Модуль `os` надає функції для взаємодії з операційною системою, зокрема, для роботи з файловою системою, а модуль `os.path` містить набір функцій для роботи з рядками (шляхами до файлів) та інші допоміжні функції для роботи з файловою системою. Функція `os.path.abspath()` повертає абсолютний шлях аргументу, з видаленням надлишкових розділювачів імен каталогів і елементів … . Функція `os.path.split()` повертає кортеж, який містить 2 елементи, перший містить шлях, а другий – ім`я файлу (який буде пустим рядком якщо ім`я файла за вказаним шляхом не задано). Ці дві частини можна отримати окремо, за допомогою функцій `os.path.dirname()` та `os.path.basename()` відповідно. Ім`я файлу також може бути розділено на дві частини – ім`я та розширення, з допомогою функції `os.path.splittext()`. Функція `os.path.join()` приймає довільну кількість рядків – шляхів і повертає єдиний шлях, з використанням розділювача каталогів, який залежить від платформи. Якщо в програмі потрібно отримати комплекс відомостей про файл або каталог, можна використовувати функцію `os.stat()`, але коли необхідно тільки окремі елементи інформації, можна використовувати відповідні функції з модуля `os.path`, наприклад, `os.path.exists()`, `os.path.getsize()`, `os.path.isfile()` або `os.path.isdir()`. Розглянемо приклад з використанням модулів `os` та `os.path`. Нижче показано, як можна використовувати модулі `os` та `os.path` для створення словника, кожен ключ якого це ім`я файлу (разом з його шляхом), а значення - мітка часу(кількість секунд, які пройшли від початку епохи), коли відбулися останні зміни файлу, для всіх файлів в каталозі `path`:

```
date_from_name = {}
for name in os.listdir(path):
    fullname = os.path.join(path, name)
    if os.path.isfile(fullname):
        date_from_name[fullname] = os.path.getmtime(fullname)
```

Цей фрагмент програмного коду доволі зрозумілий, але він може використовуватися тільки для створення переліку файлів в одному каталозі. За необхідності виконати обхід всього дерева каталогів, можна скористатися функцією `os.walk()`. Наступний фрагмент програми створює словник, кожен ключ котрого це кортеж з двох елементів(розмір файлу та ім`я файлу), а ім`я файлу не містить шлях до нього. Кожне значення словника - це список повних імен файлів, які відповідають імені файлу в ключі та мають такий самий розмір:

```
data = collections.defaultdict(list)
for root, dirs, files in os.walk(path):
    for filename in files:
        fullname = os.path. join(root, filename)
        key = (os.path.getsize(fullname), filename)
        data[key].append(fullname)
```

Для кожного каталогу функція `os.walk()` повертає шлях до кореневого каталогу піддерева та два списки, один з них – це список підкаталогів в каталозі, а другий – список файлів в каталозі. Щоб отримати повний шлях до файлу, необхідно об’єднати шлях до кореня та ім`я файлу. Після накопичення всієї необхідної інформації виконується обхід словника і виводиться звіт про можливі дублікати файлів:

```
for size, filename in sorted(data):
    names = data[(size, filename)]
    if len(names) > 1:
        print("{0} ({1} bytes) may be duplicated "
              "({2} files):".format(filename, size, len(names)))
    for name in names:
        print("\t{0} ".format(name))
```

Модуль `shutil` для потреб, пов'язаних з файлами та директоріями, надає інтерфейс більш високого рівня ніж `os`, що спрощує програмування:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db') # копіювання
>>> shutil.move('/build/executables', 'installdir') # переміщення
```

Модуль `shutil` надає функції для роботи с файлами і каталогами. `shutil.copy()` та `shutil.copytree()`, дозволяють копіювати файли і цілі дерева каталогів; `shutil.move()`, дозволяє переміщувати дерева каталогів, а `shutil.rmtree()`, дозволяє видаляти цілі дерева каталогів, навіть якщо вони пусті.

Тимчасові файли та каталоги повинні створюватися за допомогою модуля `tempfile`, який містить всі необхідні для цього функції, наприклад, `temptile.mkstemp()`, та забезпечує максимально можливу безпеку тимчасових файлів.

Модуль `filecmp` може використовуватися для порівняння файлів з допомогою функції `filecmp.cmp()` та цілих каталогів - з допомогою функції `filecmp.cmpfiles()`.

Модуль `calendar` дозволяє працювати з календарем. Серед Функцій модуля `calendar` можна виділити:

`calendar.setfirstweekday(weekday)` - встановлює перший день тижня (0 - понеділок, 1 – вівторок, … 6 - неділя). Ці значення визначені у змінних `calendar.MONDAY, calendar.TUESDAY, calendar.WEDNESDAY, calendar.THURSDAY, calendar.FRIDAY, calendar.SATURDAY` та `calendar.SUNDAY`.
`calendar.firstweekday()` – повертає перший день тижня.
`calendar.isleap(year)` – повертає булеве значення, яке дозволяє встановити чи рік є високосним.
`calendar.leapdays(y1, y2)` – кількість високосних років від y1 до y2.
`calendar.weekday(year, month, day)` - день тижня для вказаної дати.
`calendar.monthrange(year, month)` - день тижня першого дня місяця та кількість днів у цьому місяці.
`calendar.monthcalendar(year, month)` – повертає представлення календаря одного місяця і вигляді списку тижнів.
Розглянемо декілька прикладів використання модуля `calendar.`

```
# Календар жовтня 2016 року
cal = calendar.monthcalendar(2016, 10)

# Календар в текстовому вигляді
cal = calendar.TextCalendar(calendar.TUESDAY)
cal_format = cal.formatmonth(2016,10,0,0)
print(cal_format)

# Календар в html cal = calendar.HTMLCalendar(calendar.TUESDAY)
cal_format = cal.formatmonth(2016,10,0,0)
print(cal_format)

# Всі вівторки кожного місяця року
for month in range(1, 13):
    cal = calendar.monthcalendar(2016, month)
    for i in cal:
        if i[calendar.TUESDAY]:
            print(i[calendar.TUESDAY], end = " ")
     print("\n")

# Всі дати проведення лабораторних робіт протягом одного місяця
def labor_days(year, month):
    cal = calendar.Calendar()
    cal = cal.monthdatescalendar(year, month)
    for week in cal:
        if week[1].month == month:
            print(week[1])
```

Модуль `datetime` містить засоби для роботи з даними, що представляють час та дату, як у складний так і в простий спосіб. Він придатний і для арифметики часових даних, хоча основна увага

приділяється ефективному добуванню даних для форматування та їхньої обробки. Модуль також має об'єкти, що розрізняють різні часові зони.

```
# створення та форматування чисел дуже просте
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2017, 10, 23)
>>> now.strftime("%m-%d-%y or %d%b %Y is a %A on the %d day of %B")
'10-23-17 or 23Oct 2017 is a Monday on the 23 day of October'

# часові дані придатні для застосування календарної арифметики
>>> birthday = date(1977, 7, 17)
>>> age = now – birthday
>>> age.days
datetime.timedelta(14708)
```

Модулі `calendar` і `datetime` містять функції і класи, призначені для роботи з датами та часом. Оскільки вони засновані на Григоріанському календарі й якщо потрібно працювати з датами в календарях до Григоріанського то ці модулі не допоможуть. Дата і час це доволі складна тема. В світі використовувались і продовжують використовуватися різні календарі. Тривалість доби не дорівнює 24 годинам, рік це не 365 днів. Використовуються також літній та зимовий час та різні часові пояси. Клас datetime.datetime (але не в класі datetime.date) забезпечує підтримку роботи з часовими поясами, хоча вона не вмикається по замовчуванню.

Модуль time використовується для роботи з мітками часу, які є простими числовими значеннями що відповідають кількості секунд, які пройшли від початку комп`ютерної епохи (1970-01-01T00:00:00вUNIX). За допомогою цього модуля можна отримати мітку поточного часу UTC (Coordinated Universal Time - універсальний глобальний час) або локального часу з врахуванням переходу на літній час, а також для створення рядків, які представляють дату, час і дату/час, в різноманітних форматах (форматовано різними способами). Також цей модуль можна використовувати для аналізу рядків, які містять дату та час.

Розглянемо приклад використання модулів `calendar`, `datetime` та `time`. Об`єкти типу `datetime.datetime` зазвичай створюються програмним способом, а об`єкти, які зберігають дату/час, отримують інформації із зовнішніх джерел. Наприклад, від часу створення файла:

```
import calendar, datetime, time
moon_datetime_a = datetime.datetime(1991, 8, 24, 18, 00, 00)
moon_time = calendar.timegm(moon_datetime_a.utctimetuple())
moon_datetime_b = datetime.datetime.utcfromtimestamp(moon_time)
moon_datetime_a.isoformat()
moon_datetime_b.isoformat()
time.strftime("%Y-%m-%dT%H:%M:%S", time.gmtime(moon_time))
```

Змінна `moon_datetime_a` це об`єкт типу `datetime.datetime` і зберігає дату та час проголошення Акту про незалежність України. Змінна `moon_time` це змінна типу int і вона зберігає кількість секунд, які пройшли з початку епохи і до незалежності України. Це число повертає функція `calendar.timegm()`, яка приймає об`єкт типу `time_struct`, який повертає функція `datetime.datetime.utctimetuple()`. В операційній системі Windows функція `datetime.datetime.utcfromtimestamp` не може працювати з від`ємними значеннями аргументу.

Змінна `moon_datetime_b` це об`єкт типу `datetime.datetime`, і її значення отримано з `moon_time`, для демонстрації можливості перетворення кількості секунд, які пройшли від початку епохи в об`єкт типу `datetime.datetime`. Останні три рядки виводять на екран однакові рядки, які містять дату в форматі ISO8601. Поточну дату/час UTC можна отримати, як об`єкт `datetime.datetime`, за допомогою виклику функції `datetime.datetime.utcnow()`, а як кількість секунд за допомогою виклику функції `time.time()`. Для отримання локальних дата/час можна використовувати `datetime.datetime.now()` або `time.mktime(time.localtime ())`.

Модуль `timeit` дозволяє оцінити продуктивність різних підходів до вирішення певної проблеми. Такою оцінкою буде швидкість виконання програми, функції чи фрагменту коду. Наприклад, можна оцінити два способи обміні значень двох змінних. Один спосіб традиційний з використанням додаткової змінної, а інший з використанням кортежів та одночасного присвоювання. Модуль `timeit` дозволяє встановити, що традиційний спосіб повільніший майже в два рази.

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.04878928914072844
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.027066160402625883
```

Література

1. https://uk.wikibooks.org/wiki/Пориньте_у_Python_3

2. Programming in Python 3 : a complete introduction to the Python language / Mark Summerfield.—2nd ed.