Тема 9. Модули sys, itertools, locale, datetime и модуль logging

I. Модуль sys

Модуль **`sys`** обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором python.

sys.argv - список аргументов командной строки, передаваемых сценарию Python. sys.argv[0] является именем скрипта (пустой строкой в интерактивной оболочке).

Примерчик вывести все аргументы командной строки:

```
for param in sys.argv:
     print (param)
```

sys.byteorder - порядок байтов. Будет иметь значение 'big' при порядке следования битов от старшего к младшему, и 'little', если наоборот (младший байт первый).

sys.builtin_module_names - кортеж строк, содержащий имена всех доступных модулей.

sys.call_tracing(функция, аргументы) - вызывает функцию с аргументами и включенной трассировкой, в то время как трассировка включена.

sys.copyright - строка, содержащая авторские права, относящиеся к интерпретатору Python.

sys._clear_type_cache() - очищает внутренний кэш типа.

sys._current_frames() - возвращает словарь-отображение идентификатора для каждого потока в верхнем кадре стека в настоящее время в этом потоке в момент вызова функции.

sys.dllhandle - целое число, определяющее дескриптор DLL Python (Windows).

sys.exc_info() - возвращает кортеж из трех значений, которые дают информацию об исключениях, обрабатывающихся в данный момент.

sys.exec_prefix - каталог установки Python.

sys.executable - путь к интерпретатору Python.

sys.exit([arg]) - ***выход из Python. Возбуждает исключение SystemExit, которое может быть перехвачено***.

sys.flags - флаги командной строки. Атрибуты только для чтения.

sys.float_info - информация о типе данных float.

sys.float_repr_style - информация о применении встроенной функции repr() для типа float.

sys.getdefaultencoding() - возвращает используемую кодировку.

sys.getdlopenflags() - значения флагов для вызовов dlopen().

sys.getfilesystemencoding() - возвращает кодировку файловой системы.

sys.getrefcount(object) - возвращает количество ссылок на объект. Аргумент функции getrefcount - еще одна ссылка на объект.

sys.getrecursionlimit() - возвращает лимит рекурсии.

sys.getsizeof(object[, default]) - возвращает размер объекта (в байтах).

sys.getswitchinterval() - интервал переключения потоков.

sys.getwindowsversion() - возвращает кортеж, описывающий версию Windows.

sys.hash_info - информация о параметрах хэширования.

sys.hexversion - версия python как шестнадцатеричное число (для 3.2.2 final это будет 30202f0).

sys.implementation - объект, содержащий информацию о запущенном интерпретаторе python.

sys.int_info - информация о типе int.

sys.intern(строка) - возвращает интернированную строку.

sys.last_type, sys.last_value, sys.last_traceback - информация об обрабатываемых исключениях. По смыслу похоже на sys.exc_info().

sys.maxsize - максимальное значение числа типа Py_ssize_t (2^{31} на 32-битных и 2^{63} на 64-битных платформах).

sys.maxunicode - максимальное число бит для хранения символа Unicode.

sys.modules - словарь имен загруженных модулей. Изменяем, поэтому можно позабавиться :)

sys.path - список путей поиска модулей.

sys.path_importer_cache - словарь-кэш для поиска объектов.

sys.platform - информация об операционной системе.

Linux (2.x and 3.x)	'linux'
Windows	'win32'
Windows/Cygwin	'cygwin'
Mac OS X	'darwin'
OS/2	'os2'
OS/2 EMX	'os2emx'

sys.prefix - папка установки интерпретатора python.

sys.ps1, sys.ps2 - первичное и вторичное приглашение интерпретатора (определены только если интерпретатор находится в интерактивном режиме). По умолчанию sys.ps1 == ">>> ", а sys.ps2 == "... ".

sys.dont_write_bytecode - если true, python не будет писать .pyc файлы.

sys.setdlopenflags(flags) - установить значения флагов для вызовов dlopen().

sys.setrecursionlimit(предел) - установить максимальную глубину рекурсии.

sys.setswitchinterval(интервал) - установить интервал переключения потоков.

sys.settrace(tracefunc) - установить "след" функции.

sys.stdin - стандартный ввод.

sys.stdout - стандартный вывод.

sys.stderr - стандартный поток ошибок.

sys.__stdin__, sys.__stdout__, sys.__stderr__ - исходные значения потоков ввода, вывода и ошибок.

sys.tracebacklimit - максимальное число уровней отслеживания.

sys.version - версия python.

sys.api_version - версия C API.

sys.version_info - Кортеж, содержащий пять компонентов номера версии.

sys.warnoptions - реализация предупреждений.

sys.winver - номер версии python, использующийся для формирования реестра Windows.

II. Модуль itertools

Модуль itertools - сборник полезных итераторов.

itertools.count(start=0, step=1)

- бесконечная арифметическая прогрессия с первым членом start и шагом step.

itertools.cycle(iterable)

- возвращает по одному значению из последовательности, повторенной бесконечное число раз.

itertools.repeat(elem, n=Inf)

- повторяет elem n раз.

itertools.accumulate(iterable) - аккумулирует суммы.

```
accumulate([1,2,3,4,5]) --> 1 3 6 10 15
```

itertools.chain(*iterables)

- возвращает по одному элементу из первого итератора, потом из второго, до тех пор, пока итераторы не кончатся.

itertools.combinations(iterable, [r])

- комбинации длиной r из iterable без повторяющихся элементов.

```
combinations('ABCD', 2) --> AB AC AD BC BD CD
```

itertools.combinations_with_replacement(iterable, r)
- комбинации длиной r из iterable с повторяющимися элементами.

```
combinations_with_replacement('ABCD', 2) -->
                    AA AB AC AD BB BC BD CC CD DD
```

itertools.compress(data, selectors) - (d[0] if s[0]), (d[1] if s[1]), ...
`compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F`

itertools.dropwhile(func, iterable)
- элементы iterable, начиная с первого, для которого func вернула ложь.
`dropwhile(lambda x: x < 5, [1,4,6,4,1]) --> 6 4 1`

itertools.filterfalse(func, iterable)
- все элементы, для которых func возвращает ложь.

itertools.groupby(iterable, key=None)
- группирует элементы по значению. Значение получается применением функции key к элементу (если аргумент key не указан, то значением является сам элемент).

```
>>>
>>> from itertools import groupby
>>> things = [("animal", "bear"), ("animal", "duck"),
                                      ("plant", "cactus"),
...                           ("vehicle", "speed boat"),
                            ("vehicle", "school bus")]
>>> for key, group in groupby(things, lambda x: x[0]):
...     for thing in group:
...         print("A %s is a %s." % (thing[1], key))
...     print()
A bear is a animal.
A duck is a animal.

A cactus is a plant.

A speed boat is a vehicle.
A school bus is a vehicle.
```

itertools.islice(iterable[, start], stop[, step])
- итератор, состоящий из среза.
-
itertools.permutations(iterable, r=None)
- перестановки длиной r из iterable.

itertools.product(*iterables, repeat=1)

- аналог вложенных циклов.

```
product('ABCD', 'xy') --> Ax Ay Bx By Cx Cy Dx Dy
```

itertools.starmap(function, iterable)

- применяет функцию к каждому элементу последовательности (каждый элемент распаковывается).

```
starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000
```

itertools.takewhile(func, iterable)

- элементы до тех пор, пока func возвращает истину.

```
takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4
```

itertools.tee(iterable, n=2)

- кортеж из n итераторов.

itertools.zip_longest(*iterables, fillvalue=None)

- как встроенная функция zip, но берет самый длинный итератор, а более короткие дополняет fillvalue.

```
zip_longest('ABCD', 'xy', fillvalue='-') -->
                                      Ax By C- D-
```

III. Модуль locale

При форматировании чисел Python по умолчанию использует англосаксонскую систему, при которой разряды целого числа отделяются друг от друга запятыми, а дробная часть от целой отделяется точкой. В континентальной Европе, например, используется другая система, при которой разряды разделяются точкой, а дробная и целая часть – запятой (https://metanit.com/python/tutorial/6.3.php):

```
# англосаксонская система
1,234.567
# европейская система
1.234,567
```

Для решения проблемы форматирования под определенную локаль в Python имеется встроенный модуль `locale`.

Для выяснения какая локаль установлена можно выполнить:

```
import local
locale.getlocale()
```

Например, вывод:

```
('Russian_Russia', '1251')
```

Для установки локали в модуле **locale** определена функция `setlocale()`. Она принимает два параметра:

```
setlocale(category, locale)
```

Первый параметр указывает на категорию, к которой применяется функция - к числам, валютам или и числам, и валютам. В качестве значения для параметра можно передавать одну из следующих констант:

- locale.LC_ALL: применяет локализацию ко всем категориям - к форматированию чисел, валют, дат и т.д.
- locale.LC_NUMERIC: применяет локализацию к числам
- locale.LC_MONETARY: применяет локализацию к валютам
- locale.LC_TIME: применяет локализацию к датам и времени
- locale.LC_CTYPE: применяет локализацию при переводе символов в верхний или нижний регистр
- locale.LC_COLLIATE: применяет локаль при сравнении строк

Второй параметр функции `setlocale` указывает на локаль, которую надо использовать.

На ОС Windows можно использовать код страны по ISO из двух символов:

для США - "us",

для Германии - "de",

для России - "ru".

Но на MacOS необходимо указывать код языка и код страны, например, для английского в США - "en_US", для немецкого в Германии - "de_DE", для русского в России - "ru_RU". По умолчанию фактически используется локаль "en_US".

Непосредственно для форматирования чисел и валют модуль `locale` предоставляет две функции:

- `currency(num)` - форматирует валюту
- `format(str, num)` - подставляет число `num` вместо плейсхолдера в строку `str`

Применяются следующие плейсхолдеры:

- `d`: для целых чисел
- `f`: для чисел с плавающей точкой
- `e`: для экспоненциальной записи чисел

Перед каждым плейсхолдером ставится знак процента %, например: `"%d"`

При выводе дробных чисел перед плейсхолдером после точки можно указать, сколько знаков в дробной части должно отображаться:

```
%.2f        # два знака в дробной части
```

Применим локализацию чисел и валют в немецкой локале:

```
import locale
```

```
locale.setlocale(locale.LC_ALL, "de") # для  Windows
# locale.setlocale(locale.LC_ALL, "de_DE")# для MacOS

number = 12345.6789
formatted = locale.format("%f", number)
print(formatted) # 12345,678900

formatted = locale.format("%.2f", number)
print(formatted) # 12345,68

formatted = locale.format("%d", number)
print(formatted) # 12345

formatted = locale.format("%e", number)
print(formatted) # 1,234568e+04

money = 234.678
formatted = locale.currency(money)
print(formatted)    # 234,68 €
```

Если вместо конкретного кода в качестве второго параметра функции `setlocale()` передаеть пустую строку, то Python будет использовать локаль, которая применяется на текущей рабочей машине.

С помощью функции `getlocale()` можно узнать текущую локаль:

```
import locale

locale.setlocale(locale.LC_ALL, "")

number = 12345.6789
formatted = locale.format("%.02f", number)
print(formatted)    # 12345,68
print(locale.getlocale())  # ('Russian_Russia', '1251')
```

IV. Модуль datetime

Основной функционал для работы с датами и временем сосредоточен в модуле datetime в виде следующих классов (по материалам https://metanit.com/python/):

Класс date

Для работы с датами воспользуемся классом `date`, который определен в модуле `datetime`. Для создания объекта `date` мы можем использовать

конструктор date, который последовательно принимает три параметра: год, месяц и день:

```
date(year, month, day)
```

Например, создадим какую-либо дату:

```
import datetime

yesterday = datetime.date(2017,5, 2)
print(yesterday)       # 2017-05-02
```

Если необходимо получить текущую дату, то можно воспользоваться методом `today()`:

```
from datetime import date

today = date.today()
print(today)       # 2017-05-03
print("{}.{}.{}".format(today.day, today.month,
                        today.year))       # 2.5.2017
```

С помощью свойств day, month, year можно получить соответственно день, месяц и год

Класс time

За работу с временем отвечает класс time. Используя его конструктор, можно создать объект времени:

```
time([hour] [, min] [, sec] [, microsec])
```

Конструктор последовательно принимает часы, минуты, секунды и микросекунды. Все параметры необязательные, и если мы какой-то параметр не передадим, то соответствующее значение будет инициализироваться нулем.

```
from datetime import time

current_time = time()
print(current_time)    # 00:00:00

current_time = time(16, 25)
print(current_time)    # 16:25:00

current_time = time(16, 25, 45)
print(current_time)    # 16:25:45
```

Класс datetime

Класс `datetime` из одноименного модуля объединяет возможности работы с датой и временем. Для создания объекта datetime можно использовать следующий конструктор:

```
datetime(year, month, day [, hour] [, min] [, sec]
                                        [, microsec])
```

Первые три параметра, представляющие год, месяц и день, являются обязательными. Остальные необязательные, и если мы не укажем для них значения, то по умолчанию они инициализируются нулем.

```
from datetime import datetime
deadline = datetime(2017, 5, 10)
print(deadline)     # 2017-05-10 00:00:00

deadline = datetime(2017, 5, 10, 4, 30)
print(deadline)     # 2017-05-10 04:30:00
```

Для получения текущих даты и времени можно вызвать метод `now()`:

```
from datetime import datetime

now = datetime.now()
print(now)     # 2017-05-03 11:18:56.239443

print("{}.{}.{}  {}:{}".format(now.day, now.month,
now.year, now.hour, now.minute))  # 3.5.2017  11:21

print(now.date())
print(now.time())
```

С помощью свойств `day, month, year, hour, minute, second` можно получить отдельные значения даты и времени. А через методы `date()` и `time()` можно получить отдельно дату и время соответственно.

Преобразование из строки в дату

Из функциональности класса datetime следует отметить метод strptime(), который позволяет распарсить строку и преобразовать ее в дату. Этот метод принимает два параметра:

```
strptime(str, format)
```

Первый параметр `str` представляет строковое определение даты и времени, а второй параметр - формат, который определяет, как различные части даты и времени расположены в этой строке.

Для определения формата можно использовать следующие коды:

- **%d**: день месяца в виде числа
- **%m**: порядковый номер месяца
- **%y**: год в виде 2-х чисел
- **%Y**: год в виде 4-х чисел
- **%H**: час в 24-х часовом формате
- **%M**: минута
- **%S**: секунда

Применим различные форматы:

```
from datetime import datetime
deadline = datetime.strptime("22/05/2017", "%d/%m/%Y")
print(deadline)     # 2017-05-22 00:00:00
```

```
deadline = datetime.strptime("22/05/2017 12:30",
"%d/%m/%Y %H:%M")
print(deadline)     # 2017-05-22 12:30:00

deadline = datetime.strptime("05-22-2017 12:30",
                                  "%m-%d-%Y %H:%M")
print(deadline)     # 2017-05-22 12:30:00
```

Операции с датами

Фоматирование дат и времени

Для форматирования объектов date и time в обоих этих классах предусмотрен метод `strftime(format)`. Этот метод принимает только один параметр, указывающий на формат, в который нужно преобразовать дату или время.

Для определения формата можно использовать один из следующих кодов форматирования:

- %a: аббревиатура дня недели. Например, Wed - от слова Wednesday (по умолчанию используются английские наименования)
- %A: день недели полностью, например, Wednesday
- %b: аббревиатура названия месяца. Например, Oct (сокращение от October)
- %B: название месяца полностью, например, October
- %d: день месяца, дополненный нулем, например, 01
- %m: номер месяца, дополненный нулем, например, 05
- %y: год в виде 2-х чисел
- %Y: год в виде 4-х чисел
- %H: час в 24-х часовом формате, например, 13
- %I: час в 12-ти часовом формате, например, 01
- %M: минута
- %S: секунда
- %f: микросекунда
- %p: указатель AM/PM
- %c: дата и время, отформатированные под текущую локаль
- %x: дата, отформатированная под текущую локаль
- %X: время, форматированное под текущую локаль

Используем различные форматы:

```
from datetime import datetime
now = datetime.now()
print(now.strftime("%Y-%m-%d"))      # 2017-05-03
print(now.strftime("%d/%m/%Y"))      # 03/05/2017
print(now.strftime("%d/%m/%y"))      # 03/05/17
```

```
print(now.strftime("%d %B %Y (%A)"))
                              # 03 May 2017 (Wednesday)
print(now.strftime("%d/%m/%y %I:%M"))  03/05/17 01:36
```

При выводе названий месяцев и дней недели по умолчанию используются английские наименования. Если мы хотим использовать текущую локаль, но то мы можем ее предварительно установить с помощью модуля `locale`:

```
from datetime import datetime
import locale
locale.setlocale(locale.LC_ALL, "ru")

now = datetime.now()
print(now.strftime("%d %B %Y (%A)"))
```

Результат выполнения скрипта:

```
06 Март 2018 (вторник)
```

Сложение и вычитани дат и времени

Нередко при работе с датами возникает необходимость добавить к какой-либо дате определенный промежуток времени или, наоборот, вычесть некоторый период. И специально для таких операций в модуле `datetime` определен класс `timedelta`. Фактически этот класс определяет некоторый период времени.

Для определения промежутка времени можно использовать конструктор `timedelta`:

```
timedelta([days] [, seconds] [, microseconds]
   [, milliseconds] [, minutes] [, hours] [, weeks])
```

В конструктор последовательно передаются дни, секунды, микросекунды, миллисекунды, минуты, часы и недели.

Определим несколько периодов:

```
from datetime import timedelta

three_hours = timedelta(hours=3)
print(three_hours)       # 3:00:00
three_hours_thirty_minutes =
             timedelta(hours=3, minutes=30)  # 3:30:00

two_days = timedelta(2)  # 2 days, 0:00:00

two_days_three_hours_thirty_minutes =
    timedelta(days=2, hours=3, minutes=30)
                                  # 2 days, 3:30:00
```

Используя `timedelta`, можно складывать или вычитать даты. Например, получим дату, которая будет через два дня:

```
from datetime import timedelta, datetime
```

```
now = datetime.now()
print(now)
two_days = timedelta(2)
in_two_days = now + two_days
print(in_two_days)
```

Вывод скрипта:

```
2018-03-06 12:20:48.524307
2018-03-08 12:20:48.524307
```

Или узнаем, сколько было времени 10 часов 15 минут назад, то есть фактически нам надо вычесть из текущего времени 10 часов и 15 минут:

```
from datetime import timedelta, datetime

now = datetime.now()
till_ten_hours_fifteen_minutes = now -
                        timedelta(hours=10, minutes=15)
print(till_ten_hours_fifteen_minutes)
```

Свойства timedelta

Класс `timedelta` имеет несколько свойств, с помощью которых мы можем получить временной промежуток:

- `days`: возвращает количество дней
- `seconds`: возвращает количество секунд
- `microseconds`: возвращает количество микросекунд

Кроме того, метод `total_seconds()` возвращает общее количество секунд, куда входят и дни, и собственно секунды, и микросекунды.

Например, узнаем какой временной период между двумя датами:

```
from datetime import timedelta, datetime

now = datetime.now()
twenty_two_may = datetime(2017, 5, 22)
period = twenty_two_may - now
print("{} дней  {} секунд   {} микросекунд".
                    format(period.days, period.seconds,
                                    period.microseconds))
# 18 дней  17537 секунд   72765 микросекунд

print("Всего: {} секунд".
                        format(period.total_seconds()))
# Всего: 1572737.072765 секунд
```

Сравнение дат

Также как и строки и числа, даты можно сравнивать с помощью стандартных операторов сравнения:

```
from datetime import datetime
```

```
now = datetime.now()
deadline = datetime(2017, 5, 22)
if now > deadline:
    print("Срок сдачи проекта прошел")
elif now.day == deadline.day and now.month ==
deadline.month and now.year == deadline.year:
    print("Срок сдачи проекта сегодня")
else:
    period = deadline - now
    print("Осталось {} дней".format(period.days))
```

V. Модуль logging

Представьте ситуацию, когда необходимо сохранить некоторые отладочные или другие важные сообщения где-нибудь, чтобы иметь возможность позже проверить, отработала ли программа, как ожидалось. Как “сохранить где-нибудь” эти сообщения?

Сделать это можно при помощи модуля **logging**. (по материалам https://wombat.org.ua/AByteOfPython/standard_library.html)

```
import os, platform, logging

if platform.platform().startswith('Windows'):
   ogging_file = os.path.join(os.getenv('HOMEDRIVE'), \
                              os.getenv('HOMEPATH'), \
                              'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                          'test.log')

print("Сохраняем лог в", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename = logging_file,
    filemode = 'w',)

logging.debug("Начало программы")
logging.info("Какие-то действия")
logging.warning("Программа умирает")
```

Вывод:

```
$ python3 use_logging.py
Сохраняем лог в C:\Users\bsu\test.log
```

Если открыть файл test.log, он будет выглядеть примерно так:

```
2018-03-06 12:31:35,033 : DEBUG : Начало программы
```

```
2018-03-06 12:31:35,033 : INFO : Какие-то действия
2018-03-06 12:31:35,033 : WARNING : Программа умирает
```

Как это работает:

Использовались три модуля из стандартной библиотеки: модуль `os` для взаимодействия с операционной системой, модуль `platform` для получения информации о платформе (т.е. операционной системе) и модуль `logging` для сохранения лога.

Прежде всего, при помощи строки, возвращаемой функцией `platform.platform()` мы проверяем, какая операционная система используется (для более подробной информации см. **`import`** `platform; help(platform))`. Если это Windows, то мы определяем диск, содержащий домашний каталог, путь к домашнему каталогу на нём и имя файла, в котором хотим сохранить информацию. Сложив все эти три части, мы получаем полный путь к файлу. Для других платформ нам нужно знать только путь к домашнему каталогу пользователя, и мы получим полный путь к файлу.

При помощи функции `os.path.join()` мы объединяем три части пути к файлу вместе. Мы используем эту функцию вместо простого объединения строк для того, чтобы гарантировать, что полный путь к файлу записан в формате, ожидаемом операционной системой.

Далее мы конфигурируем модуль `logging` таким образом, чтобы он записывал все сообщения в определённом формате в указанный файл.

Наконец, мы можем выводить сообщения, предназначенные для отладки, информирования, предупреждения и даже критические сообщения. После выполнения программы можно просмотреть этот файл и узнать, что происходило в программе, хотя пользователю, запустившему программу, ничего не было показано.