**Тема занятия №11: Работа с датой и временем**

**1. Введение**

Python содержит несколько инструментов, которые разработчик может использовать для работы с датой и временем. Мы изучим принципы их работы и способы их применения. Начнем с модуля datetime.

**Модуль datetime**

Мы изучим следующие классы модуля:

datetime.date

datetime.timedelta

datetime.datetime

Благодаря этим классам мы в будущем сможем работать с большей частью случаев, в которых нам понадобятся объекты date и datetime. Также существует класс tzinfo, который применяется для работы с часовыми поясами. Для дополнительной информации об этом классе вы можете обратиться к документации Python.

**2. Модуль datetime**

**datetime.date**

Python может представлять даты различными способами. Для начала, рассмотрим формат datetime.date, так как это один из самых простых объектов date.

**print( datetime.date(2012, 13, 14) )**

**Traceback (most recent call last):**

**File "<string>", line 1, in <fragment>**

**builtins.ValueError: month must be in 1..12**

**print(datetime.date(2012, 12, 14)) # datetime.date(2012, 12, 14)**

В данном коде показано, как создать простой объект даты. Класс date принимает три аргумента: год, месяц и день. Если вы укажите неправильное значение, вы увидите ошибку ValueError, как в случае выше. В противном случае вы увидите, что объект datetime.date вернулся. Давайте взглянем на еще один пример:

**import datetime**

**d = datetime.date(2012, 12, 14)**

**print(d.year) # 2012**

**print(d.day) # 14**

**print(d.month) # 12**

Здесь мы присваиваем объекту date переменную d. Теперь мы можем получить доступ к различным компонентам даты по названиям, таким как d.year или d.month. Давайте посмотрим, какой сейчас день:

**import datetime**

**print(datetime.date.today()) # datetime.date(2017, 4, 5)**

Это может быть полезно, когда вам нужно записать, какой сейчас день. Или, если вам нужно выполнить основанный на сегодняшней дате расчет. Мелочь, а приятно.

**datetime.datetime**

Объект datetime.datetime содержит всю информацию объектов datetime.date плюс datetime.time. Давайте приведем несколько примеров, для лучшего понимания разницы между этим объектом, и объектом datetime.date.

**import datetime**

**a = datetime.datetime(2017, 3, 5)**

**print(a) # datetime.datetime(2017, 3, 5, 0, 0)**

**b = datetime.datetime(2017, 3, 5, 12, 30, 10)**

**print(b) # datetime.datetime(2017, 3, 5, 12, 30, 10)**

**d = datetime.datetime(2017, 3, 5, 12, 30, 10)**

**print(d.year) # 2017**

**print(d.second) # 10**

**print(d.hour) # 12**

Мы видим, что datetime.datetime принимает несколько дополнительных аргументов: год, месяц, день, час, минута и секунда. Это также позволяет вам указывать информацию о микросекундах и часовом поясе. При работе с базами данных, данные типы объектов будут использоваться достаточно часто. Большую часть вашей работы, вам нужно будет конвертировать форматы date или datetime Python в форматы SQL datetime или timestamp. Обратите внимание на то, что today совместно с datetime.datetime использует два разных метода:

**import datetime**

**a = datetime.datetime.today()**

**print(a) # datetime.datetime(2017, 4, 5, 0, 16, 54, 989663)**

**b = datetime.datetime.now()**

**print(b) # datetime.datetime(2017, 4, 5, 0, 17, 8, 24239)**

Модуль datetime содержит другой метод, под названием strftime. Этот метод позволяет разработчику создавать строку, отображающую время в более понятной для человека форме. Давайте взглянем на несколько примеров, показывающих всю полезность данного метода:

**import datetime**

**a = datetime.datetime.today().strftime("%Y%m%d")**

**print(a) # '20170405'**

**today = datetime.datetime.today()**

**print( today.strftime("%m/%d/%Y") ) # '04/05/2017'**

**print( today.strftime("%Y-%m-%d-%H.%M.%S") ) # 2017-04-05-00.18.00**

Первый пример – это скорее хитрость. В нем показано, как конвертировать сегодняшний объект datetime в строку, следующую за форматом YYYYMMDD (ГГГГММДД). Второй пример более наглядный.

В нем мы присваиваем объект datetime переменной под названием today и применяем два разных параметра форматирования строки. Первый параметр добавляет косые черточки между элементами datetime, а также перегруппировывает datetime, теперь он делится на месяц, день и год. В последнем примере мы создаем временную отметку, которая следует типичному формату: YYYY-MM-DD.HH.MM.SS. Если вам нужно указать год как двухзначный (“YY”), вы можете заменить %Y на %y.

**datetime.timedelta**

Объект datetime.timedelta отображает длительность времени. Другими словами, это разница между двумя датами или временными отметками. Давайте взглянем на наглядный пример:

**import datetime**

**# Значение: datetime.datetime(2017, 4, 5, 0, 18, 51, 980187)**

**now = datetime.datetime.now()**

**then = datetime.datetime(2017, 2, 26)**

**# Кол-во времени между датами.**

**delta = now - then**

**print(delta.days) # 38**

**print(delta.seconds) # 1131**

Мы создали два объекта datetime. Один указывает на сегодняшний день, второй – на прошедшую неделю. После этого, мы берем разницу между ними. Это возвращает объект timedelta, который мы можем далее использовать, чтобы выяснить, сколько прошло дней или секунд, между этими двумя датами. Если вам нужно узнать количество часов или минут между двумя датами, вам понадобится немножко математики, чтобы выяснить это. Давайте взглянем на проверенный способ:

**seconds = delta.total\_seconds()**

**hours = seconds // 3600**

**print(hours) # 186.0**

**minutes = (seconds % 3600) // 60**

**print(minutes) # 13.0**

Отсюда мы узнаем, что в неделе 186 часов и 13 минут. Обратите внимание на то, что мы используем двойную косую черту в качестве нашего оператора деления, также известного как floor division. Теперь мы готовы к тому, чтобы узнать больше о модуле time Python.

Модуль time открывает разработчику Python доступ к нескольким связанным со временем функциям. Модуль основан на «эпохе», точке, с которой начинается время. Для систем Unix, эпоха началась в 1970 году. Чтобы узнать, какая эпоха в вашей системе, попробуйте запустить следующий код:

**import time**

**print(time.gmtime(0))**

**Результат**

**time.struct\_time(tm\_year=1970, tm\_mon=1, tm\_mday=1, tm\_hour=0, tm\_min=0, tm\_sec=\**

**0, tm\_wday=3, tm\_yday=1, tm\_isdst=0)**

Запустил его на Windows 7, которая также уверена в том, что начало времен датируется 1970м годом. В любом случае, в данном разделе мы ознакомимся со следующими функциями:

**time.ctime**

**time.sleep**

**time.strftime**

**time.time**

**Приступим!**

**time.ctime**

Функция time.ctime конвертирует время в секунды, начиная с эпохи, в строку, показывающую местное время. Если вы ничего не передадите данной функции, то настоящее время вернется обратно. Давайте взглянем на несколько примеров:

**import time**

**print(time.ctime()) # 'Wed Apr 5 00:02:49 2017'**

**print(time.ctime(1384112639)) # 'Sun Nov 10 13:43:59 2013'**

В данном коде мы показали результаты вызова функции time.ctime, как со случайным набором секунд, начиная с эпохи, так и с пустым значением. Это можно применить, к примеру, в том случае, кода кому-нибудь нужно сохранить дату в секундах, начиная с эпохи, после чего конвертировать это в более удабриваемый формат. Немного проще, сохранить большое целое число (или длинное) в базу данных, чем страдать над ним, форматируя объект datetime в какой-либо объект даты, который принимает база данных. Конечно, это также имеет свой недостаток: вам, возможно, нужно будет преобразовать целое число или значение с плавающей запятой обратно в строку.

**time.sleep**

Функция time.sleep дает разработчику возможность приостановить выполнение его скрипта на определенное количество секунд. Это можно сравнить с добавлением функции «Пауза» в вашу программу. Я нашел этот класс особенно полезным, когда мне нужно было подождать несколько секунд, пока закроется файл, или база данных закончит выполнять свою задачу. Давайте взглянем на пример. Откройте новое окно в IDLE и сохраните следующий код:

**import time**

**for x in range(5):**

**time.sleep(2)**

**print("Slept for 2 seconds")**

Теперь запустите этот код в IDLE. Сделав это, вы увидите фразу «Slept for 2 seconds» пять раз, с интервалом в 2 секунды между каждым сообщением. Это очень легко в использовании!

**time.strftime**

Модуль time имеет функцию strftime, которая работает по схожему с версией datetime принципу. Разница заключается, главным образом, в том, что именно она принимает для ввода: кортеж, или объект struct\_time, вроде тех, которые возвращаются после вызова time.gmtime() или time.localtime(). Вот небольшой пример:

**import time**

**a = time.strftime("%Y-%m-%d-%H.%M.%S", time.localtime())**

**print(a) # '2017-04-05-00.11.20'**

Этот код очень похож на код временной отметки, который мы создавали в разделе datetime. На мой взгляд, метод datetime более интуитивен при создании объекта datetime.datetime а затем при вызове его метода strftime в том формате, который вам более удобен. С модулем time, вам нужно передать формат и кортеж времени. На самом деле выбор за вами, каким именно способом вам будет удобнее решить вашу задачу.

**time.time**

Функция time.time отображает время в секундах, начиная с эпохи, как число с плавающей запятой. Давайте взглянем:

**import time**

**x = time.time()**

**print(x) # 1491340367.478385**

Весьма просто. Вы можете использовать данную функцию, когда вам нужно сохранить настоящее время в базу данных, но вам не нужно конвертировать его в метод datetime вашей базы данных. Вы можете вспомнить о том, что метод ctime принимает время в секундах, так что мы можем использовать time.time для получения количества секунд для передачи их ctime, вот так:

**import time**

**a = time.ctime(time.time())**

**print(a) # Wed Apr 5 00:13:47 2017**

**3. Получение текущей даты и времени**

Для вывода даты и времени нужно выполнить следующее:

**import datetime**

**day\_time = datetime.datetime.now()**

**print(day\_time)**

**Получение текущей даты и времени в Python datetime**

Как видно мы получаем для создания объекта даты и времени используется класс с тем же именем что и модуль datetime.

Когда нужно получить только дату используется класс date:

**import datetime**

**day = datetime.date.today()**

**print(day)**

Получение только времени выполняется через метод time():

**import datetime**

**date\_time = datetime.datetime.now().time()**

**print(date\_time, type(date\_time))**

Каждый из описываемых классов можно импортировать следующим способом:

**from datetime import date**

**from datetime import datetime**

**print(datetime.now())**

**print(date.today())**

**4. Форматирование и перевод в строку**

Для получения части даты или времени можно использовать следующие атрибуты:

year

month

day

weekday

hour

minute

second

**import datetime**

**# Получение дня от даты**

**date = datetime.date.today()**

**print(date.day)**

**date\_time = datetime.datetime.now()**

**# Получение часа суток от времени**

**print(date\_time.hour)**

Так же есть метод strftime, который форматирует даты в нужном формате в строку.

Например, так мы получим дату в формате, который используется у нас:

**import datetime**

**date = datetime.datetime.today()**

**print(date.strftime('%d-%m-%Y %H:%M'))**

Где:

%d - день месяца с 1 по 31;

%m - месяц с 1 по 12;

%Y - год;

%H - час в формате 0-24;

%M - минуты;

%S - секунды.

Таким же способом можно получить время и дату:

%c - время и дата;

%x - дата;

%X - время.

date = datetime.datetime.today()

print(date.strftime('%x'))

Обратите внимание, что таким способом мы преобразуем объект класса datetime в строку и мы больше не сможем использовать методы по работе с датой (например сравнение):

**import datetime**

**date = datetime.datetime.today()**

**local\_date = date.strftime('%d-%m-%Y %H:%M')**

**print('Объект класса datetime: ', type(date))**

**print('Дата преобразованная в строку: ', type(local\_date))**

**print(local\_date.day)**

Мы получим ошибку так как уже работаем со строкой:

**AttributeError: 'str' object has no attribute 'day'**

Выше описаны основные возможности форматирования используя метод strftime(), но их, конечно, больше.

**5. Получения дня недели и название месяца**

Можно получить название дня недели или название. Численный вариант эквивалентен следующим значениям:

0 - Monday (Понедельник);

1 - Tuesday (Вторник);

2 - Wednesday (Среда);

3 - Thursday (Четверг);

4 - Friday (Пятница);

5 - Saturday (Суббота);

6 - Sunday (Воскресенье).

Следующий пример вернет день недели в виде числа:

**date = datetime.datetime.today()**

**print(date.weekday())**

Или получить название:

**date = datetime.datetime.today()**

**print(date.strftime('%A'))**

**print(date.strftime('%a'))**

**Получение имени дня недели в Python datetime**

Где:

%A - полное название дня недели;

%a - сокращенное название дня недели;

%s - представление в виде числа.

Такой же принцип по работе с месяцами, где:

%B - полное название месяца;

%b - сокращенное название месяца;

%m - месяц в виде числа.

**Создание объекта даты и времени**

Для создания даты используется класс с аргументами date(год, месяц, число):

**date = datetime.date(2019, 1, 13)**

**print(date, type(date))**

Можно создать дату и время. В обоих случаях мы можем использовать именованные аргументы если так удобнее:

**date\_time = datetime.datetime(year = 2019,**

**month = 2,**

**day = 4,**

**hour = 7,**

**minute = 9,**

**second = 13 )**

**print(date\_time, type(date\_time))**

Каждый аргумент времени, по умолчанию, имеет значение 0. Мы так же можем использовать подход выше для получения, например, только года или времени:

**date\_time = datetime.datetime(year = 2019,**

**month = 2,**

**day = 4,**

**second = 13 )**

**# Получаем только время**

**time = date\_time.strftime('%X')**

**print(time, type(time))**

**date = datetime.date(**

**year = 2018,**

**month = 5,**

**day = 3**

**)**

**# Получаем только год**

**year = date.year**

**print(year, type(year))**

Если вы не укажете год, месяц или день, то получите ошибку т.к. они по умолчанию равны None:

**TypeError: Required argument 'year' (pos 1) not found**

**Есть отдельный класс для создания времени time:**

**time = datetime.time(12, 33, second=33, microsecond=122)**

**print(time, type(time))**

**print(time.hour)**

Используя strptime() можно создавать объект datetime из строки:

**date\_string = '21 September. 1999'**

**date\_object = datetime.strptime(date\_string, '%d %B. %Y')**

**print(date\_object, type(date\_object))**

timedelta - это класс с помощью которого можно установить не дату, как в примерах выше, а продолжительность. Так мы создадим объект с 12 днями и 33 секундами:

**duration\_time = datetime.timedelta(days = 12, seconds = 33)**

**print(duration\_time, type(duration\_time))**

**Продолжительность с timedelta в Python datetime**

Все атрибуты, которые мы можем указывать для этого класса:

days

seconds

microseconds

milliseconds

minutes

hours

Кроме этого мы можем преобразовывать эти объекты в секунды:

**duration\_time = datetime.timedelta(minutes = 1, seconds = 10)**

**# Преобразуем в секунды**

**seconds = duration\_time.total\_seconds()**

**print(seconds, type(seconds))**

**6. Разница между датами**

Мы можем искать разницу между датами получая объект timedelta:

**date\_now = datetime.datetime.now()**

**date\_tomorrow = datetime.datetime(2019, 12, 29)**

**# Вычисляем разницу между датами**

**difference = date\_tomorrow - date\_now**

**print(difference, type(difference))**

**print('Разница только в днях: ', difference.days)**

Каждый их объектов выше можно изменить. Так мы изменим объект timedelta прибавив часы к минутам:

**duration\_hours = datetime.timedelta(hours = 1)**

**duration\_minutes = datetime.timedelta(minutes = 10)**

**result = duration\_hours + duration\_minutes**

**print(result, type(result))**

С помощью timedelta изменяется и дата. Пример ниже изменяет текущую дату прибавляя к ней 1 день и 1 час:

**date\_now = datetime.datetime.now()**

**duration\_minutes = datetime.timedelta(days=1, minutes=10)**

**result = date\_now + duration\_minutes**

**print(result, type(result))**

**7. Сравнение и условия**

Каких-либо хитростей в сравнении объектов нет. В следующем примере идет сравнение двух дат с разницей в день:

**date\_one = datetime.datetime(2017,12,1)**

**date\_two = datetime.datetime(2017,12,2)**

**if date\_two > date\_one:**

**print('{0} больше чем {1}'.format(date\_two, date\_one))**

**Сравнение объектов datetime в Python**

При этом стоит проверять, что объекты относятся к классу datetime, а не являются строками. В следующем примере мы сравниваем года, но они уже относятся к численным:

**date\_one = datetime.datetime(2017,12,1)**

**date\_two = datetime.datetime(2017,12,2)**

**if date\_two.year == date\_one.year:**

**print('Год одинаковый. Тип данных: ', type(date\_two.year))**

Объект timedelta тоже можно сравнивать:

**time\_one = datetime.timedelta(seconds=2)**

**time\_two = datetime.timedelta(minutes=1)**

**if time\_one < time\_two:**

**print('Pass')**

**8. Работа с метками (штампами) timestamp**

При работе с API или в Unix системах мы можем увидеть отображение времени в таком формате:

1578238360

Данные числа обозначают количество секунд с 1 января 1970 года. Мы можем конвертировать данное число в понятный формат используя datetime:

**timestamp = date.fromtimestamp(1578238360)**

**print(timestamp)**

Так же сработает если мы хотим получить и время:

**timestamp = datetime.fromtimestamp(1578238360)**

**print(timestamp)**

Для конвертирования в timestamp используется метод с аналогичным названием:

**from datetime import datetime**

**date\_now = datetime.now()**

**print(date\_now.timestamp)**

**Работа с часовыми поясами**

Библиотека datetime не хранит часовые пояса, данные о переводах часов (летнее и зимнее время) и високосных секундах. К тому же, некоторые страны, могут изменить время опираясь на локальные ситуации. Эти ситуации опасны, когда идет запись в базу данных. Для вывода в GUI, можно использовать datetime.datetime.now() или высчитывать часовой пояс из базы.

Для записи в базу данных мы можем использовать время UTC и отдельно считать часовой пояс:

**utc = datetime.datetime.utcnow()**

**print(utc)**

**Время UTC в Python Datetime**

Следующий пример вычислит разницу времени между UTC и локальным.

**gmt = datetime.today() - datetime.utcnow()**

**print(gmt)**

Для вычисления других часовых поясов можно использовать стороннюю библиотеку pytz, которая их хранит:

pip install pytz

Вывести все часовые пояса мы можем так:

**import pytz**

**for tz in pytz.all\_timezones:**

**print(tz)**

На примере ниже я перевожу локальное время в часовой пояс Киева:

**import pytz**

**from datetime import datetime**

**tz\_kiev = pytz.timezone('Europe/Kiev')**

**time\_kiev = datetime.now(tz\_kiev)**

**print(time\_kiev)**