**Модуль 12. Работа с датами**

[Оценка времени выполнения кода в языке Python 1](#_Toc148659823)

[Работа с датами на примере работы модуля datetime 2](#_Toc148659824)

[Класс date 3](#_Toc148659825)

[Класс time 3](#_Toc148659826)

[Класс datetime 4](#_Toc148659827)

[Преобразование из строки в дату 5](#_Toc148659828)

[Фоматирование дат и времени 5](#_Toc148659829)

[Сложение и вычитание дат и времени 7](#_Toc148659830)

[Свойства timedelta 8](#_Toc148659831)

[Сравнение дат 9](#_Toc148659832)

[Модули сalendar и tkcalendar 9](#_Toc148659833)

# Оценка времени выполнения кода в языке Python

Одним из важных аспектов при разработке программного обеспечения является оптимизация производительности. Оценка времени выполнения кода играет важную роль в определении эффективности алгоритмов и выявлении мест, где можно улучшить скорость работы программы. Одним из способов данной оценки является модуль time. Модуль time предоставляет функции для работы со временем в Python. Одна из таких функций — time.time(), которая возвращает количество секунд, прошедших с начала эпохи (обычно 1 января 1970 года). Можно использовать эту функцию для измерения времени выполнения определенного участка кода:

 В Python есть несколько способов измерения времени выполнения, которые помогут вам в этом процессе. В этой статье мы рассмотрим некоторые из них с примерами кода.

import time  
  
start\_time = time.time()  
*# Ваш код, время выполнения которого нужно измерить*end\_time = time.time()  
execution\_time = end\_time - start\_time  
print(f"Время выполнения: {execution\_time} секунд")

Модуль timeit предназначен специально для измерения времени выполнения небольших фрагментов кода. Он предоставляет функцию timeit(), которая автоматически повторяет выполнение кода несколько раз и возвращает среднее время выполнения:

from timeit import timeit  
  
t = timeit('print("Hello from Specialist")')  
print(t)

миллион раз выполнит функцию print

или

t = timeit('"-".join(str(n) for n in range(100))', number=10000)  
  
print(t)

import timeit  
  
code\_to\_test = """  
# Код, время выполнения которого нужно измерить  
"""  
execution\_time = timeit.timeit(code\_to\_test, number=1000)  
print(f"Среднее время выполнения: {execution\_time} секунд")

from timeit import Timer

def get\_fib\_recurs(n):  
 if n == 1 or n == 2:  
 return 1  
 else:  
 return get\_fib(n - 1) + get\_fib(n - 2)  
  
  
def get\_fib(n):  
 a, b = 1, 1  
 i = 2  
 while i < n:  
 a, b = b, a + b  
 i += 1  
 return b  
  
t = Timer('get\_fib(20000)', "from \_\_main\_\_ import get\_fib")  
print(t.timeit(number=200))  
  
t = Timer('get\_fib\_recurs(20000)', "from \_\_main\_\_ import get\_fib\_recurs")  
print(t.timeit(number=200))

# Работа с датами на примере работы модуля datetime

Основной функционал для работы с датами и временем сосредоточен в модуле datetime в виде следующих классов:

date

time

datetime

## 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 :

## 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(2023, 5, 10)  
print(deadline) *# 2023-05-10 00:00:00*deadline = datetime(2023, 5, 10, 4, 30)  
print(deadline) *# 2023-05-10 04:30:00*

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

now = datetime.now()  
print(now)  
  
print(**f"{**now.day**}.{**now.month**}.{**now.year**} {**now.year**}:{**now.minute**}"**)

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.

from datetime import datetime  
  
birthday = datetime(1978, 12, 18)  
date\_now = datetime.now()  
  
age = date\_now - birthday  
sec = age.total\_seconds()  
  
min, \_ = divmod(sec, 60)  
print(min, 'мин')  
  
hours, \_ = divmod(min, 60)  
print(hours, 'часов')  
  
days, \_ = divmod(hours, 24)  
print(days, 'дней')  
  
years, days = divmod(days, 365)  
print(years, 'лет', days, 'дней')

Результат в консоли:

23633263.0 мин

393887.0 часов

16411.0 дней

44.0 лет 351.0 дней

Или age = date\_now - birthday  
print(age.days//365)

yesterday = datetime.date(2023, 12, 31)  
print(yesterday)

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

from datetime import datetime

birthday = date(1978, 12, 18)

print(date\_now.year)  
print(date\_now.month)  
print(date\_now.day)

birthday = date(1978, 12, 18)

print(birthday.year)  
print(birthday.month)  
print(birthday.day)

date\_now = date.today()

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

Из секунд получить дату, время

t0 = time.time()  
print(t0)  
print(datetime.fromtimestamp(t0))

## 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

**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*

# 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 мы можем 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:

deadline = datetime.strptime(**"22/05/2024"**, **"%d/%m/%Y"**)  
print(deadline)   
  
deadline = datetime.strptime(**"22/05/2025 12:30"**, **"%d/%m/%Y %H:%M"**)  
print(deadline)  
  
deadline = datetime.strptime(**"05-22-2026 12:30"**, **"%m-%d-%Y %H:%M"**)  
print(deadline)

## 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 мы можем 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:

now = datetime.now()  
print(now.strftime(**"%Y-%m-%d"**))   
print(now.strftime(**"%d/%m/%Y"**))   
print(now.strftime(**"%d/%m/%y"**))   
print(now.strftime(**"%d %B %Y (%A)"**))   
print(now.strftime(**"%d/%m/%y %I:%M"**))

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

**import** locale  
  
locale.setlocale(locale.LC\_ALL, **""**)  
  
now = datetime.now()  
print(now.strftime(**"%d %B %Y (%A)"**))

# Сложение и вычитание 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 мы последовательно передаем дни, секунды, микросекунды, миллисекунды, минуты, часы и недели.

Определим несколько периодов:

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*print(two\_days)  
  
two\_days\_three\_hours\_thirty\_minutes = timedelta(days=2, hours=3, minutes=30) *# 2 days, 3:30:00*print(two\_days\_three\_hours\_thirty\_minutes)

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

now = datetime.now()  
print(now)  
two\_days = timedelta(2)  
in\_two\_days = now + two\_days  
print(in\_two\_days)

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))

# Модули сalendar и tkcalendar

Пример

**import** calendar

print(calendar.month(2023, 7))

Мы можем распечатать календарь на целый год, используя следующую функцию: calendar.calendar(yyyy, w, l, c, m)

calendar.isleap(2016)  
calendar.leapdays(year1, year2)

Возврат дня недели

Метод буднего дня принимает 3 аргумента, а именно: год, месяц и день. Функция возвращает день недели, при этом понедельник имеет индекс 0, а воскресенье – 6.

Например: calendar.weekday(2019, 3, 21) Вывод: 3

Как видно, эта функция возвращает значение индекса «3», то есть «четверг». Получение сокращенных названий дней недели

Функция weekheader принимает аргумент n, который указывает количество символов для определенного имени дня недели и возвращает заголовок, содержащий сокращенные имена дней недели.

Например: print (calendar.weekheader(2))

Вывод: Mo Tu We Th Fr Sa Su

По аналогии: print (calendar.weekheader(3))

Вывод: Mon Tue Wed Thu Fri Sat Sun

Получение количества дней в месяце

Функция monthrange принимает 2 аргумента: год и месяц.

Эта функция возвращает кортеж, содержащий индекс дня недели, в который начинается месяц, и количество дней в месяце.

Например: print (calendar.monthrange(1983, 12)) Вывод: {3,31}

Поскольку первый день декабря 1983 г. был четвергом, функция возвращает значение индекса четверга в качестве первого элемента кортежа и 31, поскольку это количество дней в декабре.

Получение недели в месяце Функция monthcalendar принимает 2 аргумента: год и месяц и возвращает матрицу, в которой каждая строка представляет неделю в этом месяце.

Например: print(calendar.monthcalendar(1983, 11))

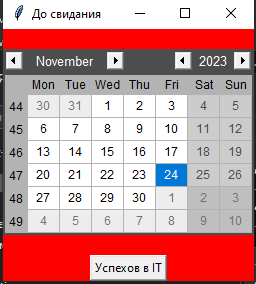
Python предлагает несколько вариантов разработки GUI (графического пользовательского интерфейса).

Из всех методов графического интерфейса чаще всего используется Tkinter. Это стандартный интерфейс Python для инструментария Tk GUI, поставляемого с Python.

Python с Tkinter — это самый быстрый и простой способ создания приложений с графическим интерфейсом.

Посмотрим, как мы можем создать калькулятор дней с даты в Tkinter, калькулятор дней с даты используется для добавления или вычитания дней из выбранной даты, чтобы получить новую дату.

Этот калькулятор используется для получения точной даты, которая наступит через заданное количество дней.

from tkinter import \*  
from tkcalendar import Calendar  
  
  
def click():  
 root.title("До свидания")  
 root["bg"] = "red"  
 btn["text"] = "Успехов в IT"  
  
  
root = Tk()  
cal = Calendar(root)  
cal.pack(pady=20)  
  
btn = Button(text="Button", command=click)  
btn.pack()  
  
root.mainloop()