

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Курганский государственный университет»

Политехнический институт

Кафедра программного обеспечения автоматизированных систем

Курсовая работа

по дисциплине «Конструирование программного обеспечения»

Направление подготовки: «Программная инженерия»

Профиль: «Технологии промышленной разработки программного обеспечения»

Группа: ИТЗ-300/18

Выполнил студент: Алексеев Е.А.

Проверил: Дик Д. И.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
СТАНДАРТЫ ОФОРМЛЕНИЯ КОДА PYTHON	5
Внешний вид кода.....	5
Пробелы в выражениях и инструкциях	9
Прочие рекомендации:	11
Комментарии	12
Имена.....	13
Стили имен	14
Общие рекомендации	15
ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ	17
ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ.....	19
ДОКУМЕНТИРОВАНИЕ КОДА	20
ВЫВОД.....	23
ЛИСТИНГ ИСХОДНОГО КОДА ПРОГРАММЫ.....	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	27

ВВЕДЕНИЕ

В большинстве перинатальных центров и профильных женских консультаций для определения срока беременности (двумя методами) применяют гестационный круг.

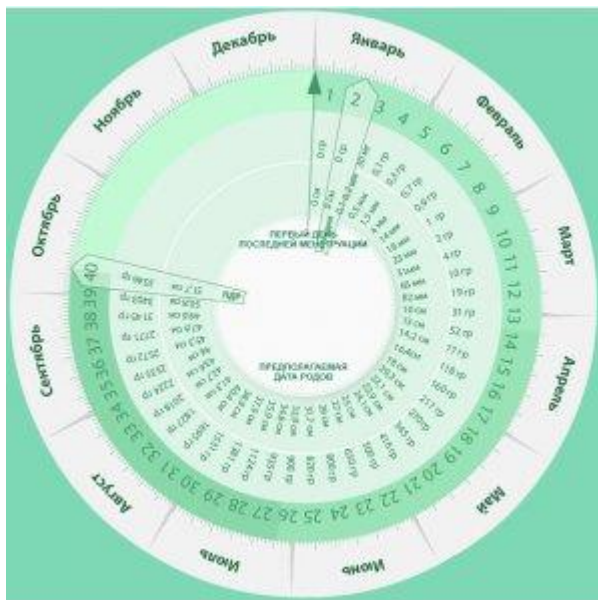


Рисунок 1 - Гестационный круг


Пример программы приведен на Рисунке 1

За многие годы он хорошо себя зарекомендовал в акушерско-гинекологической практике. Однако, автоматизация рабочих процессов в здравоохранении затронула и его. Поскольку врач заполняет протоколы осмотра в электронном виде, гораздо проще автоматизировать и расчёт сроков предполагаемой беременности акушерским и календарным методом. Для этого было разработано веб-приложение на языке Python 3. Оно позволяет более эффективно использовать время приёма пациента и исключает человеческий фактор при расчёте, т.к. теперь за результат отвечает приложение.

Интерфейс разработанного приложения представлен на рисунке 2.

Калькулятор срока беременности:

Укажите дату последней менструации:

Укажите дату прохождения УЗИ:

Укажите срок беременности по УЗИ:


Рассчитать

Рисунок 2 - Калькулятор срока беременности

После заполнения необходимых для расчёта данных, в нижнем поле приложения получаем результат, как на рисунке 3.

Калькулятор срока беременности:

Укажите дату последней менструации:

Укажите дату прохождения УЗИ:

Укажите срок беременности по УЗИ:

Рассчитать

Акушерский срок составляет: 19 нед. 0 дн.
Предполагаемая дата родов: 08.11.2021
Дата родов по УЗИ: 23.10.2021

Рисунок 3 - Результат работы программы

РФ КГУ 09.03.04 КР21 16123

СТАНДАРТЫ ОФОРМЛЕНИЯ КОДА PYTHON

Внешний вид кода

Отступы

Используйте 4 пробела на каждый уровень отступа.

Продолжительные строки должны выравнивать обернутые элементы либо вертикально, используя неявную линию в скобках (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа следует применять следующие соображения: на первой линии не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение линии.

Пробелы и табуляция

Пробелы - самый предпочтительный метод отступов.

Табуляция должна использоваться только для поддержки кода, написанного с отступами с помощью табуляции.

Python 3 запрещает смешивание табуляции и пробелов в отступах.

Python 2 пытается преобразовать табуляцию в пробелы.

Когда вы вызываете интерпретатор Python 2 в командной строке с параметром `-t`, он выдает предупреждения (warnings) при использовании смешанного стиля в отступах, а запустив интерпретатор с параметром `-tt`, вы получите в этих местах ошибки (errors). Эти параметры очень рекомендуются!

Максимальная длина строки

Ограничьте длину строки максимум 79 символами.

Для более длинных блоков текста с меньшими структурными ограничениями (строки документации или комментарии), длину строки следует ограничить 72 символами.

Ограничение необходимой ширины окна редактора позволяет иметь несколько открытых файлов бок о бок, и хорошо работает при использовании инструментов анализа кода, которые предоставляют две версии в соседних столбцах.

Некоторые команды предпочитают большую длину строки. Для кода, поддерживающегося исключительно или преимущественно этой группой, в которой могут прийти к согласию по этому вопросу, нормально увеличение длины строки с 80 до 100 символов (фактически увеличивая максимальную длину до 99 символов), при условии, что комментарии и строки документации все еще будут 72 символа.

Стандартная библиотека Python консервативна и требует ограничения длины строки в 79 символов (а строк документации/комментариев в 72).

Предпочтительный способ переноса длинных строк является использование подразумеваемых продолжений строк Python внутри круглых, квадратных и фигурных скобок. Длинные строки могут быть разбиты на несколько строк, обернутые в скобки. Это предпочтительнее использования обратной косой черты для продолжения строки.

Пустые строки

Отделяйте функции верхнего уровня и определения классов двумя пустыми строками.

Определения методов внутри класса разделяются одной пустой строкой.

Дополнительные пустые строки возможно использовать для разделения различных групп похожих функций. Пустые строки могут быть опущены между несколькими связанными однострочниками (например, набор фиктивных реализаций).

Используйте пустые строки в функциях, чтобы указать логические разделы.

Python расценивает символ `control+L` как незначащий (whitespace), и вы можете использовать его, потому что многие редакторы обрабатывают его как

разрыв страницы — таким образом логические части в файле будут на разных страницах. Однако, не все редакторы распознают control+L и могут на его месте отображать другой символ.

Кодировка исходного файла

Кодировка Python должна быть UTF-8 (ASCII в Python 2).

Файлы в ASCII (Python 2) или UTF-8 (Python 3) не должны иметь объявления кодировки.

В стандартной библиотеке, нестандартные кодировки должны использоваться только для целей тестирования, либо когда комментарий или строка документации требует упомянуть имя автора, содержащего не ASCII символы; в остальных случаях использование `\x`, `\u`, `\U` или `\N` - наиболее предпочтительный способ включить не ASCII символы в строковых литералах.

Начиная с версии python 3.0 в стандартной библиотеке действует следующее соглашение: все идентификаторы обязаны содержать только ASCII символы, и означать английские слова везде, где это возможно (во многих случаях используются сокращения или неанглийские технические термины). Кроме того, строки и комментарии тоже должны содержать лишь ASCII символы. Исключения составляют: (а) test case, тестирующий не-ASCII особенности программы, и (б) имена авторов. Авторы, чьи имена основаны не на латинском алфавите, должны транслитерировать свои имена в латиницу.

Проектам с открытым кодом для широкой аудитории также рекомендуется использовать это соглашение.

Import-секции

Каждый импорт, как правило, должен быть на отдельной строке.

Правильно:

```
import os
import sys
```

Неправильно:

```
import sys, os
```

В то же время, можно писать так:

```
from subprocess import Popen, PIPE
```

Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант.

Импорты должны быть сгруппированы в следующем порядке:

импорты из стандартной библиотеки

импорты сторонних библиотек

импорты модулей текущего проекта

Вставляйте пустую строку между каждой группой импортов.

Указывайте спецификации `__all__` после импортов.

Рекомендуется абсолютное импортирование, так как оно обычно более читаемо и ведет себя лучше (или, по крайней мере, даёт понятные сообщения об ошибках) если импортируемая система настроена неправильно (например, когда каталог внутри пакета заканчивается на `sys.path`):

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

Тем не менее, явный относительный импорт является приемлемой альтернативой абсолютному импорту, особенно при работе со сложными пакетами, где использование абсолютного импорта было бы излишне подробным:

```
from . import sibling
from .sibling import example
```

В стандартной библиотеке следует избегать сложной структуры пакетов и всегда использовать абсолютные импорты.

Неявные относительно импорты никогда не должны быть использованы, и были удалены в Python 3.

Когда вы импортируете класс из модуля, вполне можно писать вот так:

```
from myclass import MyClass
from foo.bar.yourclass import YourClass
```

Если такое написание вызывает конфликт имен, тогда пишете:

```
import myclass
import foo.bar.yourclass
```

И используйте "myclass.MyClass" и "foo.bar.yourclass.YourClass".

Шаблоны импортов (from import *) следует избегать, так как они делают неясным то, какие имена присутствуют в глобальном пространстве имён, что вводит в заблуждение как читателей, так и многие автоматизированные средства. Существует один оправданный пример использования шаблона импорта, который заключается в опубликовании внутреннего интерфейса как часть общественного API (например, переписав реализацию на чистом Python в модуле акселератора (и не будет заранее известно, какие именно функции будут переписаны)).

Пробелы в выражениях и инструкциях

Избегайте использования пробелов в следующих ситуациях:

Непосредственно внутри круглых, квадратных или фигурных скобок.

Правильно:

```
spam(ham[1], {eggs: 2})
```

Неправильно:

```
spam( ham[ 1 ], { eggs: 2 } )
```

Непосредственно перед запятой, точкой с запятой или двоеточием:

Правильно:

```
if x == 4: print(x, y); x, y = y, x
```

Неправильно:

```
if x == 4 : print(x , y) ; x , y = y , x
```

Сразу перед открывающей скобкой, после которой начинается список аргументов при вызове функции:

Правильно:

```
spam(1)
```

Неправильно:

```
spam (1)
```

Сразу перед открывающей скобкой, после которой следует индекс или срез:

Правильно:

```
dict['key'] = list[index]
```

Неправильно:

```
dict ['key'] = list [index]
```

Использование более одного пробела вокруг оператора присваивания (или любого другого) для того, чтобы выровнять его с другим:

Правильно:

```
x = 1
y = 2
long_variable = 3
```

Неправильно:

```
x           = 1
y           = 2
long_variable = 3
```

Прочие рекомендации:

Всегда окружайте эти бинарные операторы одним пробелом с каждой стороны: присваивания (`=`, `+=`, `-=` и другие), сравнения (`==`, `<`, `>`, `!=`, `<>`, `<=`, `>=`, `in`, `not in`, `is`, `is not`), логические (`and`, `or`, `not`).

Если используются операторы с разными приоритетами, попробуйте добавить пробелы вокруг операторов с самым низким приоритетом. Используйте свои собственные суждения, однако, никогда не используйте более одного пробела, и всегда используйте одинаковое количество пробелов по обе стороны бинарного оператора.

Правильно:

```
i = i + 1
submitted += 1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Неправильно:

```
i=i+1
submitted +=1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Не используйте пробелы вокруг знака `=`, если он используется для обозначения именованного аргумента или значения параметров по умолчанию.

Правильно:

```
def complex(real, imag=0.0):
    return magic(r=real, i=imag)
```

Неправильно:

```
def complex(real, imag = 0.0):
    return magic(r = real, i = imag)
```

Не используйте составные инструкции (несколько команд в одной строке).

Правильно:

```
if foo == 'blah':
    do_blah_thing()
do_one()
do_two()
do_three()
```

Неправильно:

```
if foo == 'blah': do_blah_thing()
do_one(); do_two(); do_three()
```

Иногда можно писать тело циклов `while`, `for` или ветку `if` в той же строке, если команда короткая, но если команд несколько, никогда так не пишите. А также избегайте длинных строк!

Точно неправильно:

```
if foo == 'blah': do_blah_thing()
for x in lst: total += x
while t < 10: t = delay()
```

Вероятно, неправильно:

```
if foo == 'blah': do_blah_thing()
else: do_non_blah_thing()

try: something()
finally: cleanup()

do_one(); do_two(); do_three(long, argument,
                             list, like, this)

if foo == 'blah': one(); two(); three()
```

Комментарии

Комментарии, противоречащие коду, хуже, чем отсутствие комментариев. Всегда исправляйте комментарии, если меняете код!

Комментарии должны являться законченными предложениями. Если комментарий — фраза или предложение, первое слово должно быть написано с большой буквы, если только это не имя переменной, которая начинается с маленькой буквы (никогда не изменяйте регистр переменной!).

Если комментарий короткий, можно опустить точку в конце предложения. Блок комментариев обычно состоит из одного или более абзацев, составленных из полноценных предложений, поэтому каждое предложение должно оканчиваться точкой.

Ставьте два пробела после точки в конце предложения.

Программисты, которые не говорят на английском языке, пожалуйста, пишите комментарии на английском, если только вы не уверены на 120%, что ваш код никогда не будут читать люди, не знающие вашего родного языка.

Имена

Соглашения об именах переменных в python немного туманны, поэтому их список никогда не будет полным — тем не менее, ниже мы приводим список рекомендаций, действующих на данный момент.

Описание: Стили имен

Существует много разных стилей. Поможем вам распознать, какой стиль именования используется, независимо от того, для чего он используется.

Обычно различают следующие стили:

- *b* (одионочная маленькая буква)
- *B* (одионочная заглавная буква)
- *lowercase* (слово в нижнем регистре)
- *lower_case_with_underscores* (слова из маленьких букв с подчеркиваниями)
- *UPPERCASE* (заглавные буквы)
- *UPPERCASE_WITH_UNDERSCORES* (слова из заглавных букв с подчеркиваниями)
- *CapitalizedWords* (слова с заглавными буквами, или *CapWords*, или *CamelCase*). Замечание: когда вы используете аббревиатуры в таком

стиле, пишите все буквы аббревиатуры заглавными — *HTTPServerError* лучше, чем *HttpServerError*.

- *mixedCase* (отличается от *CapitalizedWords* тем, что первое слово начинается с маленькой буквы)
- *Capitalized_Words_With_Underscores* (слова с заглавными буквами и подчеркиваниями — уродливо!)

Стили имен

Имена, которых следует избегать

Никогда не используйте символы *l* (маленькая латинская буква «эль»), *O* (заглавная латинская буква «о») или *I* (заглавная латинская буква «ай») как однобуквенные идентификаторы.

В некоторых шрифтах эти символы неотличимы от цифры один и нуля (и символа вертикальной палочки, — прим. перев.) Если очень нужно использовать *l* имена, пишите вместо неё заглавную *L*.

Имена модулей и пакетов

Модули должны иметь короткие имена, состоящие из маленьких букв. Можно использовать и символы подчеркивания, если это улучшает читабельность. То же, за исключением символов подчеркивания, относится и к именам пакетов.

Имена классов

Все имена классов должны следовать соглашению *CapWords* почти без исключений. Классы внутреннего использования могут начинаться с символа подчеркивания.

Имена глобальных переменных

Будем надеяться, что такие имена используются только внутри одного модуля. Руководствуйтесь теми же соглашениями, что и для имен функций.

Имена функций

Имена функций должны состоять из маленьких букв, а слова разделяться символами подчеркивания — это необходимо, чтобы увеличить читабельность.

Аргументы функций и методов

Если имя аргумента конфликтует с зарезервированным ключевым словом `python`, обычно лучше добавить в конец имени символ подчеркивания, чем исказить написание слова или использовать аббревиатуру. Таким образом, `print_` лучше, чем `prnt`.

Общие рекомендации

- Код должен быть написан так, чтобы не зависеть от разных реализация языка (PyPy, Jython, IronPython, Pyrex, Psycο и пр.).
- Пользуйтесь `".startswith()"` и `".endswith()"` вместо обработки частей строк (*string slicing*) для проверки суффиксов или префиксов. `startswith()` и `endswith()` выглядят чище и порождают меньше ошибок.

Например:

Правильно:

```
if foo.startswith('bar'):
```

Неправильно:

```
if foo[:3] == 'bar':
```

1. Для последовательностей (строк, списков, кортежей) можно использовать тот факт, что пустая последовательность есть *false*:

Правильно:

```
if not seq:
    if seq:
```

Неправильно:

```
if len(seq):
    if not len(seq):
```

2. Не сравнивайте логические типы с *True* и *False* с помощью `==`:

Правильно:

```
if greeting:
```

Неправильно:

```
if greeting == True:
```

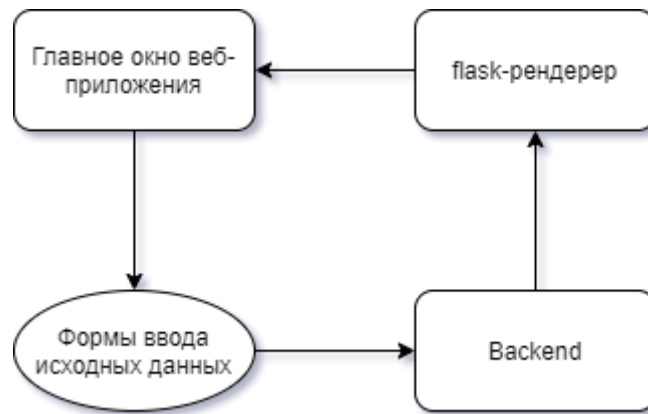

ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Рисунок 4 - Схема работы веб-приложения

При запуске приложения (пользователям сделаны ярлыки на рабочих столах, а также закладки в браузерах) загружается интерфейс главной страницы приложения. (Рисунок 2 – Введение).

Пользователь приложения (в нашем случае врач или медицинская сестра) выполняют ввод исходных данных пациентки и нажимают кнопку

«Рассчитать». Пример интерфейса после заполнения исходных данных можно увидеть на рисунке 4.

Калькулятор срока беременности:

Укажите дату последней менструации:

01.02.2021



Укажите дату прохождения УЗИ:

15.03.2021



Укажите срок беременности по УЗИ:

9

2

Рассчитать

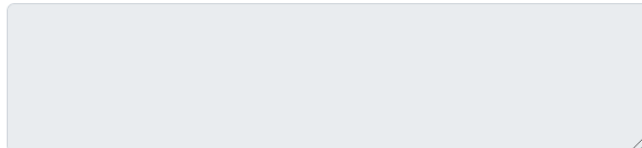


Рисунок 5 - Пример ввода исходных данных в приложение

Далее пользователь жмёт на кнопку «Рассчитать» и получает результат (Пример в разделе «Введение» на рисунке 3) в сером блоке программы. Он защищён от ручного ввода данных и позволяет просматривать исключительно результат расчёта.

РФ КГУ 09.03.04 КР21 16123

ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ

Веб-приложение собирает данные, введенные пользователем посредством специальных HTML-форм, содержимое которых записывается в переменные и вычисляется «на лету», т.е. без промежуточного сохранения и дальнейшего хранения в какой-либо СУБД. Результат выводится пользователю в процессе повторного рендеринга страницы с выводом в соответствующую форму. В ней можно как просто просмотреть результат, так и скопировать его в буфер обмена устройства, с которого выполняется работа. При повторном вычислении старые данные затираются новыми, т.е. хранение не предусмотрено.

РФ КГУ 09.03.04 КР21 16123

ДОКУМЕНТИРОВАНИЕ КОДА

Calc

[Титульная страница](#) [Пакеты ▾](#) [Файлы ▾](#)

Функции | Переменные

Пространство имен app

Функции

datetime	make_date_from_str (str date_string, str template="%Y-%m-%d") Функция перевода даты из str в datetime-объект Подробнее...
timedelta	count_pregnacy_time (datetime start_date, datetime current_date=datetime.now()) Функция для расчёта срока беременности Подробнее...
str	data_to_weeks (timedelta pregnancy_delta) Функция для перевода срока беременности в формат недели/дни Подробнее...
str	period (str str_date) Функция для вывода даты в неделях Подробнее...
def	index () Функция для генерации index.html. Подробнее...
def	handle_data () Расчёт параметров, рендер HTML-шаблона Подробнее...

Переменные

app = Flask(__name__)
port = int(os.environ.get("PORT", 5000))
host

Рисунок 6 - Функции

Функции

◆ **count_pregnacy_time()**

```
timedelta app.count_pregnacy_time ( datetime start_date,
                                   datetime current_date = datetime.now()
                                   )
```

Функция для расчёта срока беременности

Аргументы

start_date прочитанная с HTML-формы дата

current_date текущая дата

Возвращает

результат расчёта дельты

◆ **data_to_weeks()**

```
str app.data_to_weeks ( timedelta pregnancy_delta )
```

Функция для перевода срока беременности в формат недели/дни

Аргументы

pregnacy_delta рассчитанный срок

Возвращает

результат в днях и неделях

Рисунок 7 - Подробное описание функций

◆ **handle_data()**

```
def app.handle_data ( )
```

Расчёт параметров, рендер HTML-шаблона

Возвращает

HTML-страница с результатом вычислений

◆ **index()**

```
def app.index ( )
```

Функция для генерации index.html.

Возвращает

HTML-страница

Рисунок 8 - Подробное описание функций

◆ **make_date_from_str()**

```
datetime app.make_date_from_str ( str date_string,
                                str template = "%Y-%m-%d"
                                )
```

Функция перевода даты из str в datetime-объект

При чтении HTML-формы приложения получаем дату в str-формате

Её необходимо перевести в читаемый формат datetime для последующей машинной обработки

Аргументы

date_string дата в строчном формате

template шаблон входных данных

Возвращает

объект даты

◆ **period()**

```
str app.period ( str str_date )
```

Функция для вывода даты в неделях

Аргументы

str_date дата в str-представлении

Возвращает

результат в неделях

Рисунок 9 - Подробное описание функций

Переменные



Рисунок 10 - Переменные

ВЫВОД

В последнее время практически все медицинские учреждения перешли на электронную форму хранения медицинских записей. Это повышает скорость оказания медицинской помощи и её качество, снижает затраты на администрирование и в целом улучшает здоровье населения.

Наряду с ростом численности персонала расширяется и спектр внедряемого в организациях здравоохранения программного обеспечения для оказания медицинской помощи и консультирования. Это очень удобный инструмент, дающий медикам быстрый доступ к медицинским данным как со стационарного компьютера, так и с мобильного устройства. Каждая новая ИТ-система рождает резкий скачок запросов на оперативное предоставление доступа к ней, даже если это необходимо лишь части медперсонала.

Калькулятор расчёта срока беременности лишь одно из огромного количества приложений, повышающих скорость и качество оказания медицинской помощи.

В процессе написания курсовой работы было изучено написание «правильного» кода, соответствующего стандартам оформления, а также его документирование для простоты дальнейшего сопровождения и внедрения в медицинских организациях.

ЛИСТИНГ ИСХОДНОГО КОДА ПРОГРАММЫ

```
import os
from datetime import datetime, timedelta
from flask import Flask, request, render_template

app = Flask(__name__)

## \brief Функция перевода даты из str в datetime-объект
## \details При чтении HTML-формы приложения получаем дату в
str-формате
## \details Её необходимо перевести в читаемый формат
datetime для последующей машинной обработки
## \param date_string дата в строчном формате
## \param template шаблон входных данных
## \return объект даты

def make_date_from_str(date_string: str, template: str =
"%Y-%m-%d") -> datetime:

    date_obj = datetime.strptime(date_string, template)

    return date_obj

## \brief Функция для расчёта срока беременности
## \param start_date прочитанная с HTML-формы дата
## \param current_date текущая дата
## \return результат расчёта дельты

def count_pregnacy_time(start_date: datetime, current_date:
datetime = datetime.now()) -> timedelta:

    pregnancy_delta = current_date - start_date

    return pregnancy_delta

## \brief Функция для перевода срока беременности в формат
недели/дни
## \param pregnancy_delta рассчитанный срок
## \return результат в днях и неделях

def data_to_weeks(pregnacy_delta: timedelta) -> str:
    weeks = pregnancy_delta.days // 7
    days = pregnancy_delta.days % 7

    result = f"{weeks} нед. {days} дн."

    return result
```



```

## \brief Функция для вывода даты в неделях
## \param str_date дата в str-представлении
## \return результат в неделях

def period(str_date: str) -> str:
    start_date = make_date_from_str(str_date)

    delta = count_pregnacy_time(start_date)

    result = data_to_weeks(delta)

    return result

@app.route('/', methods=['GET', 'POST'])
## \brief Функция для генерации index.html
## \return HTML-страница
def index():
    return render_template('index.html')

@app.route('/calc', methods=['GET', 'POST'])
## \brief Расчёт параметров, рендер HTML-шаблона
## \return HTML-страница с результатом вычислений
def handle_data():
    mdt = request.form['MDT']
    udt = request.form['UDT']
    a = make_date_from_str(mdt)
    b = a + timedelta(weeks=40)
    c = period(mdt)
    d = datetime.date(b)
    e = d.year
    f = d.month
    g = d.day
    h = f'{g:02d}.{f:02d}.{e:02d}'
    i = make_date_from_str(udt)
    j = request.form['USN']
    k = request.form['USD']
    k1 = 7 * int(j)
    m = 280 - (int(k) + int(k1))
    n = i + timedelta(days=m)
    o = n.year
    p = n.month
    q = n.day
    r = f'{q:02d}.{p:02d}.{o:02d}'

    if int(j) == 0 and int(k) == 0:
        r = h
    else:
        pass

    if int(j) < 0 or int(k) < 0:

```

```
        r = h
    else:
        pass

    def_text = f'Акушерский срок составляет: {c}
\nПредполагаемая дата родов: {h} \nДата родов по УЗИ: {r}'
    return render_template('index.html', def_text=def_text)

if __name__ == '__main__':
    port = int(os.environ.get('PORT', 5000))
    app.run(host='0.0.0.0', port=port)
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. PEP 8 - руководство по написанию кода на Python, <https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html>;
2. Совершенный код. Мастер-класс | Макконнелл Стив, 2017 г.;
3. Чистый код. Создание, анализ и рефакторинг | Мартин Роберт К., 2010.
4. Интернет-ресурс <https://refactoring.guru/ru>;
5. Интернет-ресурс <https://www.doxygen.nl/manual/index.html>.