**Базовая страница для проекта: BasePage**

Давайте перепишем тест из файла *test\_main\_page.py* с помощью паттерна Page Object, который мы добавили на этапе [подготовки окружения.](https://stepik.org/lesson/199980/step/6?unit=174035) Мы будем работать с главной страницей нашего приложения, поэтому дадим классу говорящее название MainPage.

**Важно!**В этом уроке мы напишем самостоятельно простую реализацию паттерна Page Object. А в следующих уроках уже рассмотрим существующие фреймворки и то, как они могут облегчить нам жизнь. Сейчас самая главная задача — осознать принципы работы.

1. Создайте в своем проекте папку pages, там мы будем хранить все наши Page Object

2. В папке создайте два файла: base\_page.py и main\_page.py

Для начала сделаем базовую страницу, от которой будут унаследованы все остальные классы. В ней мы опишем вспомогательные методы для работы с драйвером.

3. В файле base\_page.py создайте класс с названием BasePage.

В Python такие вещи делаются с помощью следующей конструкции:

class BasePage():

4. Теперь в наш класс нужно добавить методы. Первым делом добавим *конструктор —*метод, который вызывается, когда мы создаем объект. Конструктор объявляется ключевым словом \_\_init\_\_. В него в качестве параметров мы передаем экземпляр драйвера и url адрес. Внутри конструктора сохраняем эти данные как аттрибуты нашего класса. Получается примерно так:

def \_\_init\_\_(self, browser, url):

self.browser = browser

self.url = url

5. Теперь добавим еще один метод open. Он должен открывать нужную страницу в браузере, используя метод get().

Объявите ниже в том же классе:

def open(self):

и реализуйте этот метод: нужна всего одна строка. Эту строку нужно отправить в качестве ответа на это задание, без отступов.

6. После того как Stepik принял ваш ответ как правильный, добавьте новые файлы в Git и зафиксируйте изменения коммитом (не забудьте осмысленное сообщение).

В итоге у вас должен следующий код в файле *base\_page.py*:

class BasePage():

def \_\_init\_\_(self, browser, url):

self.browser = browser

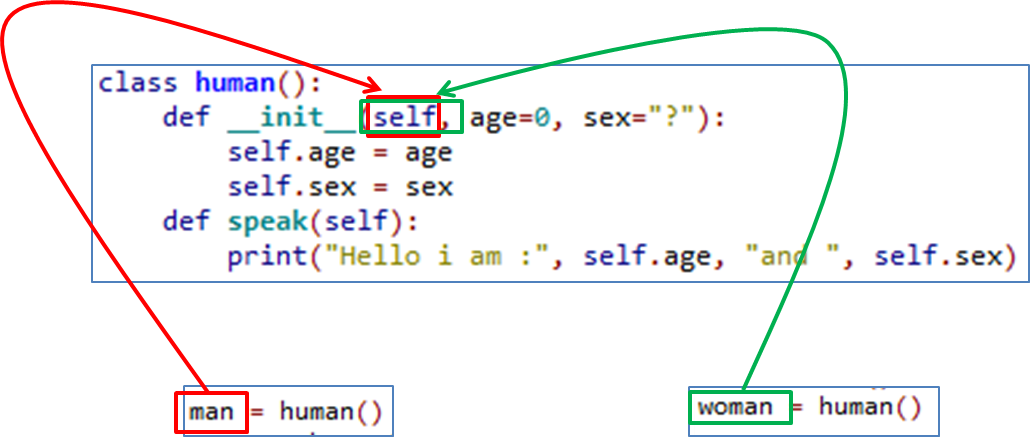
self.url = url

def open(self):

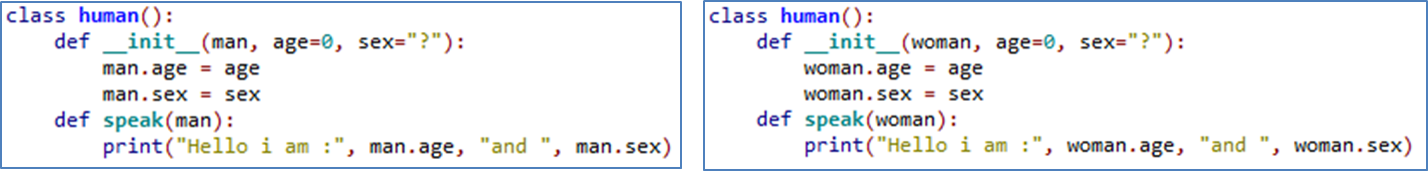
self.browser.get(self.url)

**SELF**

When objects are instantiated, the object itself is passed into the self parameter.



Because of this, the object’s data is bound to the object. Below is an example of how you might like to visualize what each object’s data might look. Notice how ‘self’ is replaced with the objects name. I'm not saying this example diagram below is wholly accurate but it hopefully with serve a purpose in visualizing the use of self.



The Object is passed into the self parameter so that the object can keep hold of its own data.

Although this may not be wholly accurate, think of the process of instantiating an object like this: When an object is made it uses the class as a template for its own data and methods. Without passing it's own name into the self parameter, the attributes and methods in the class would remain as a general template and would not be referenced to (belong to) the object. So by passing the object's name into the self parameter it means that if 100 objects are instantiated from the one class, they can all keep track of their own data and methods.

See the illustration below:



## Page Object для главной страницы сайта

Теперь реализуем Page Object, который будет связан с главной страницей интернет-магазина.

1. Откройте файл main\_page.py

2. В нем нужно сделать импорт базового класса BasePage:

from .base\_page import BasePage

3. В нем создайте класс  MainPage. Его нужно сделать наследником класса BasePage. Класс-предок в Python указывается в скобках:

class MainPage(BasePage):

таким образом, класс MainPage будет иметь доступ ко всем атрибутам и методам своего класса-предка.

4. Перенесите метод из предыдущего урока в класс MainPage:

def go\_to\_login\_page(browser):

   login\_link = browser.find\_element(By.CSS\_SELECTOR, "#login\_link")

login\_link.click()

Чтобы все работало, надо слегка видоизменить его. В аргументы больше не надо передавать экземпляр браузера, мы его передаем и сохраняем на этапе создания Page Object. Вместо него нужно указать аргумент self , чтобы иметь доступ к атрибутам и методам класса:

def go\_to\_login\_page(self):

Так как браузер у нас хранится как аргумент класса BasePage, обращаться к нему нужно соответствующим образом с помощью self:

self.browser.find\_element(By.CSS\_SELECTOR, "#login\_link")

Итого, файл main\_page.py будет выглядеть так:

from .base\_page import BasePage

from selenium.webdriver.common.by import By

class MainPage(BasePage):

def go\_to\_login\_page(self):

login\_link = self.browser.find\_element(By.CSS\_SELECTOR, "#login\_link")

login\_link.click()

## Проверка элемента на странице

Чтобы выводить адекватное сообщение об ошибке, мы будем все проверки осуществлять с помощью assert и перехватывать исключения.

Для этого напишем вспомогательный метод поиска элемента в нашей базовой странице BasePage, который будет возвращать нам True или False. Можно сделать это по-разному (с настройкой явных или неявных ожиданий). Сейчас воспользуемся неявным ожиданием.

1. В конструктор BasePage добавим команду для неявного ожидания со значением по умолчанию в 10:

def \_\_init\_\_(self, browser, url, timeout=10):

self.browser = browser

self.url = url

self.browser.implicitly\_wait(timeout)

2. Теперь в этом же классе реализуем метод is\_element\_present, в котором будем перехватывать исключение. В него будем передавать два аргумента: как искать (css, id, xpath и тд) и собственно что искать (строку-селектор).

Чтобы перехватывать исключение, нужна конструкция try/except:

def is\_element\_present(self, how, what):

try:

self.browser.find\_element(how, what)

except (имя исключения):

return False

return True

Чтобы импортировать нужное нам исключение, в самом верху файла нужно указать:

from selenium.common.exceptions import имя\_исключения

Отлично! Теперь для всех проверок, что элемент действительно присутствует на странице, мы можем использовать этот метод.

3. Теперь модифицируем метод проверки ссылки на логин так, чтобы он выдавал адекватное сообщение об ошибке:

def should\_be\_login\_link(self):

assert self.is\_element\_present(By.CSS\_SELECTOR, "#login\_link\_invalid"), "Login link is not presented"

Запустите тесты и посмотрите, что вывод об ошибке стал более понятным:

pytest -v --tb=line --language=en test\_main\_page.py

**И не забудьте заменить селектор на правильный, чтобы тест снова проходил!**