Предлагаю зарегистрироваться на бесплатном курсе от Яндекса по изучению языка Питон

https://practicum.yandex.ru/promo/python/dev?utm_source=yandex&utm_medium=cpc&utm_campaign=Yan_Sch_RF_Pyth_Des_Intro_460&utm_content=sty_search%3As_none%3Acid_56602826%3Agid_45231386963%3Apid_3079590442 6%3Aaid_12329063931%3Acrid_0%3Arid_%3Ap_1%3Apty_premium%3Amty_syn%3Amkw_%3Adty_desktop%3Acgcid_0%3Arn_Республика%20Крым%3Arid_977&utm_term=циклы%20в%20питоне&_openstat=ZGlyZWNoLnlhbmRleC5ydTs1NjYwMjgyNjsxMjMyOTA2MzkzMTt5YW5kZXgucnU6cHJlbWl1bQ&yclid=11395853289821831167

Там можно параллельно проходить изучение и выполнять задания

Цикл for

Как было сказано ранее, мы используем **цикл** в тех случаях, когда вам нужно повторить что-нибудь n-ное количество раз. Это проще понять, если взглянуть на пример. Мы используем встроенную функцию Python range. Функция range создаст список длинной в «n» элементов. В Python версии 2.X существует другая функция под названием **xrange**, которая является генератором чисел и не такая ресурсоемкая, как **range**. Ранее разработчики **сменили xrange на range** в Python 3. Вот пример:

```
Python
1print(range(5)) # ответ: range(0, 5)
```

Как вы видите, функция **range** взяла целое число и вернула **объект range**. Функция range также принимает начальное значение, конечное значение и значение шага. Вот еще два примера:

```
Python
1a = range(5, 10)
2print(a) # range(5, 10)
3
4b = list(range(1, 10, 2))
5print(b) # [1, 3, 5, 7, 9]
```

В пером примере показано, что вы можете передать **начальное** и **конечное** значение, и функция range вернет числа, начиная с начального значения вплоть до (но не включая) последнее значение. Например, при запросе 5-10 мы получим 5-9. Во втором примере видно, как использовать функцию списка (list) для того, чтобы **функция range** вернула каждый второй элемент, между 1 и 10. Так что она начинает с 1, пропускает 2 и так далее. Теперь вы, наверное, гадаете, что же именно она будет делать с **циклами**? Что-ж, есть один простой способ показать, как работает цикл с использованием **функции range**! Давайте взглянем:

```
Python
1for number in range(5):
2    print(number)
```

Что здесь произошло? Давайте почитаем слева на право, чтобы понять это. Для каждого числа в диапазоне 5 мы вводим число. Мы знаем, что если мы вызываем range со значением 5, мы получим список из 5 элементов. Так что каждый раз, проходя через цикл, она выводит каждый из элементов. Цикл for, показанный выше, может быть эквивалентом следующего:

```
Python
1for number in [0, 1, 2, 3, 4]:
2    print(number)
```

Функция **range** лишь делает результат несколько меньшим. **Цикл for** может обходить любой итератор Python. Мы уже видели, как именно он может работать со списком. Давайте взглянем, может ли он выполнять итерацию со словарем.

```
Python
a_dict = {"one":1, "two":2, "three":3}

for key in a_dict:
    print(key)
```

Когда вы используете **for в словаре**, вы увидите, что он автоматически **перебирает ключи**. Вам не нужно указывать ключ for в **a_dict.keys()** (впрочем, это также работает). Python делает только нужные нам вещи. Вы возможно думаете, почему ключи выводятся в другом порядке, отличном от того, какой был указан в словаре? Как мы знаем из соответствующей статьи, словари не упорядочены, так что мы можем выполнять итерацию над ними, при этом ключи могут быть в любом порядке. Теперь, зная, что ключи могут быть **отсортированы**, вы можете отсортировать их до итерации. Давайте немного изменим словарь, чтобы увидеть, как это работает.

```
Python
a_dict = {1:"one", 2:"two", 3:"three"}
keys = a_dict.keys()

keys = sorted(keys)
for key in keys:
    print(key)
```

Результат:

```
Python
1
2
3
```

Давайте остановимся и разберемся с тем, что делает этот код. Во-первых, мы создали словарь, в котором ключи выступают в качестве целых чисел, вместо строк. Далее, мы **извлекли ключи из словаря**. Каждый раз, когда вы взываете метод **keys**(), он возвращает неупорядоченный список ключей. Если вы выведите их, и увидите, что они расположен в порядке по возрастанию, то это просто случайность. Теперь у нас есть доступ к ключам словаря, которые хранятся в переменной, под названием **keys**. Мы **сортируем наш список**, после чего используем цикл for в нем. Теперь мы готовы к тому, чтобы сделать все немного интереснее. Мы попробуем применить **цикл в функции range**, но нам нужно вывести только целые числа. Чтобы сделать это, нам нужно использовать условный оператор вместо параметра шага range. Это можно сделать следующим образом:

```
Python
for number in range(10):
    if number % 2 == 0:
        print(number)
```

Результат:

```
Python
0
2
4
```

```
46
58
```

Вы наверное гадаете, что вообще здесь происходит? Что еще за знак процента? В Python, % называется **оператором модуля**. Когда вы используете оператор модуля, он **возвращает остаток**. Когда вы делите целое число на два, вы получаете число без остатка, так что мы выводим эти числа. Вам, возможно, не захочется использовать оператор модуля часто в будущем, но в моей работе он нередко помогает.

Рачем нужны условные инструкции

Сила и важность условных операторов в том, что большая часть логики описывается именно с их помощью. Они делают одну простую, но очень важную вещь: говорят интерпретатору Питона какую часть кода выполнять (какую не выполнять) в зависимости от какого-либо условия. Эти конструкции можно вкладывать друг в друга. Это называется логическим ветвлением. Однако, стоит проявлять особую осторожность с вложенностью: каждый новый уровень вложенности увеличивает сложность кода. Сложнее читать (понимать), выше вероятность допустить ошибку.

Как работает if else

Синтаксис

Условный оператор в Python следует широко распространённому в других языках программирования синтаксису: if else. Самый простой пример:

```
КОПИРОВАТЬ

if True:
    print("Всё правильно")# Вывод:


Всё правильно
```

Давайте рассмотрим, что же здесь произошло:
1. Сперва указывается if с которого начинается оператор.
2. После if указывается условие. Если значение истинно, то выполняется код, следующий в «теле» оператора. В данном случае мы, вместо условия, указали сразу True (истина).
3. Двоеточие после условия обязательно.
4. Тело оператора – инструкции, которые будут выполнены в случае истинности условия.

У этого оператора есть и расширенные возможности. Если вам нужно выполнить другой код, но только если условие ложно, используйте else:

```
КОПИРОВАТЬ

if 1 == 2:
```

```
   print("Всё правильно")else:

   print("Да нет же!")# Вывод:
```

```
Да нет же!
```

В теле оператора может быть сколько угодно строк:

```
КОПИРОВАТЬ
```

```
i = 0if i == 0:

   print("Число равно нулю")

   print("И на него нельзя делить")else:

   print("На такое число делить можно")

   print("Так как оно не равно нулю")# Вывод:
```

```
Число равно нулю


И на него нельзя делить
```

Другими словами, выполнение программы доходит до проверки условия в операторе, а затем перенаправляется в необходимый блок кода, который будет выполняться строка за строкой до самого конца блока. Остальные блоки оператора игнорируются.

Отступы

Одной из отличительных черт языка Python является использование отступов для выделения блоков кода. Такой подход имеет ряд преимуществ и главным из них является визуальная чистота. Отступы представляют из себя пробелы или табуляции в начале строк:

```
КОПИРОВАТЬ
```

```
# начало кода# код# код# код
```

```
# первый блок кода

# код

# код

    # второй блок кода

    # код

    # код

    # конец второго блока

# конец первого блока# продолжение основного кода
```

Если отступы одинакового размера и следуют строка за строкой, интерпретатор языка идентифицирует их как единый программный блок.

КОПИРОВАТЬ

```
# Начало основного потока# выполнения программы

яблоки = 5

груши = 10if яблоки < груши:

  # Входим в первый блок

  print('Яблок меньше чем груш')

  if груши > 7:

    # Входим во второй блок

    print('И груш больше семи')

    print('Варим компот из груш')

  else:

    # Входим в третий блок
```

```
    print('Но груш меньше семи')

    print('Варим компот из яблок и груш')else:

  # Входим в четвёртый блок

  print('Груш меньше чем яблок')

  print('Варим компот из яблок')# Вывод:
```

Яблок меньше чем груш

И груш больше семи

Варим компот из груш

И так, варьируя ширину отступов, мы можем создавать различные структуры вложенности блоков.

Примеры

Давайте рассмотрим несколько примеров из реального, «боевого» кода:

Пример №1: класс для парсинга интернет-страниц:

```
КОПИРОВАТЬ


# coding=utf-8from pathlib import Path

from selenium import webdriverfrom selenium.webdriver.common.by import Byfrom selenium.webdriver.common.desired_capabilities import DesiredCapabilitiesfrom selenium.webdriver.firefox.options import Optionsfrom selenium.webdriver.support import expected_conditions as ecfrom selenium.webdriver.support.wait import WebDriverWait



class ObjSel:
```

```
  def __init__(self):
      pass


  def __enter__(self):
      return self


  def init(self, adress, headless=True):
      options = Options()
      setattr(options, 'headless', headless)
      caps = DesiredCapabilities().FIREFOX
      caps["pageLoadStrategy"] = 'normal'  # Ждём полной загрузки страницы. Если не хотим ждать- >
"eager"
      self.driver = webdriver.Firefox(capabilities=caps,
      firefox_options=options,
      executable_path=Path('geckodriver.exe'), )
      self.vars = {}
      self.driver.implicitly_wait(10)
      self.driver.get(adress)


  def web_manager(self, name, manager_type='element'):
      if name is None:
          print('Wrong name')
      if manager_type == 'element':
```

```
        if name.get('By.XPATH'):
            return self.driver.find_element(By.XPATH, name['By.XPATH'])
        elif name.get('By.CSS_SELECTOR'):
            return self.driver.find_element(By.CSS_SELECTOR, name['By.CSS_SELECTOR'])
    elif manager_type == 'clickable':
        if name.get('By.XPATH'):
            return WebDriverWait(self.driver, 3).until(
            ec.element_to_be_clickable((By.XPATH, name['By.XPATH'])))
        elif name.get('By.CSS_SELECTOR'):
            return WebDriverWait(self.driver, 3).until(
            ec.element_to_be_clickable((By.CSS_SELECTOR, name['By.CSS_SELECTOR'])))
    elif manager_type == 'elements':
        if name.get('By.XPATH'):
            return self.driver.find_elements(By.XPATH, name['By.XPATH'])
        elif name.get('By.CSS_SELECTOR'):
            return self.driver.find_elements(By.CSS_SELECTOR, name['By.CSS_SELECTOR'])


def __exit__(self, exc_type, exc_val, exc_tb):
    if 'driver' in self.__dict__.keys():
        print("Удаление экземпляра Selenium")
        self.driver.quit()
```

Здесь Вы можете увидеть несколько примеров использования условных операторов.
Пример №2: ниже приведена функция модуля queue из стандартной библиотеки. Эта функция получает значение из очереди:

КОПИРОВАТЬ

```
def get(self, block=True, timeout=None):



  with self.not_empty:
      if not block:
          if not self._qsize():
              raise Empty
      elif timeout is None:
          while not self._qsize():
              self.not_empty.wait()
      elif timeout < 0:
          raise ValueError("'timeout' must be a non-negative number")
      else:
          endtime = time() + timeout
          while not self._qsize():
              remaining = endtime - time()
              if remaining <= 0.0:
                  raise Empty
              self.not_empty.wait(remaining)
      item = self._get()
```

```
        self.not_full.notify()

        return item
```

Оператор elif

Логика бывает не только бинарной, но и множественной. К примеру, что если дальнейшее выполнение программы зависит от одного из четырёх вариантов? Во многих языках программирования (особенно, старых) принято эту задачу решать так:

```
КОПИРОВАТЬ


i = 4if i == 1:

   print("Один")else:

   if i == 2:

      print("Два")

   else:

      if i == 3:

         print("Три")

      else:

         if i == 4:

            print("Четыре")# Вывод:



Четыре
```

Но, такой код, как мы уже говорили, создаёт сложности в дальнейшем сопровождении из-за глубоких уровней вложенности. К счастью, Пайтон предоставляет более совершенный инструмент: elif. Перепишем предыдущий код:

```
КОПИРОВАТЬ


i = 4if i == 1:
```

```
  print("Один")elif i == 2:

  print("Два")elif i == 3:

  print("Три")elif i == 4:

  print("Четыре")# Вывод:



Четыре
```

Так получается нагляднее, не так ли?
Этих условий может быть не ограниченное количество. Они перебираются по очереди, пока одно из условий не окажется верным. После этого поток управления передаётся в тело elif, а после его выполнения выходит из условного оператора.

Заглушка pass
В теле условного оператора обязательно что-то должно быть указано. Если Вы хотите пропустить какое-то условие, Вы можете использовать оператор-заглушку pass.

Такой подход чаще всего используют при разработке на стадии прототипирования.

```
КОПИРОВАТЬ

глаза = "голубые"

характер = "мягкий"if глаза == "карие":

  print('Познакомлюсь')elif глаза == "голубые":

  if характер == "мягкий":

    print('Женюсь!')

  else:

    passelse:

  pass# Вывод:



Женюсь!
```

if else в одну строку

В Python предусмотрена возможность записывать условный оператор в одну строку:

```
КОПИРОВАТЬ

Что делать если Да if Условие else Что делать если Нет
```

Пример:

```
КОПИРОВАТЬ

стакан = 1print("Быть") if стакан / 2 >= 0.5 else print("Не быть")# Вывод:



Быть
```

Но эту конструкцию стоит использовать только для простых условий и действий. Почему? Если её использовать для более сложных вещей, читаемость кода очень сильно падает. Как Вы думаете, что выведет следующий код:

```
КОПИРОВАТЬ

x = 15print(x // 7) if x % 3 >= 5 else print(x) if x % 4 < 2 else print(x * -1)# Вывод:



?
```

Конструкция switch case

На момент написания этого урока, всё сообщество питонистов ждёт выход релиза Python 3.10. В этой версии, кроме прочего, появится, хорошо знакомое программистам из других языков, структурное сопоставление с шаблоном или switch/case. В Питоне же эта конструкция будет выглядеть как match/case. Сопоставление подразумевает определение при операторе match искомого значения, после которого можно перечислить несколько потенциальных кейсов, каждый с оператором case. В месте обнаружения совпадения между match и case выполняется соответствующий код. Пример из документации:

```
КОПИРОВАТЬ

match x:
```

```
case host, port:


    mode = "http"

case host, port, mode:


    pass
```