MLClass. "Прикладной анализ данных".

Курс "Инструментарий Data Science".

Преподаватель ФКН НИУ ВШЭ Кашницкий Юрий

Урок 2. Основы языка Python

Часть 5. Циклы

Цикл While

Цикл While очень удобен, потому что он позволяет Вашей программе работать, пока Вы не решите её остановить. Эта конструкция позволяет организовать бесконечный цикл, который ждет действий пользователя.

Что такое цикл while?

Цикл while проверяет начальные условия. Если условие выполняется, цикл начинает работать. Каждый раз когда проходит одна итерация, условие проверяется снова. До тех пор, пока условие выполняется, цикл продолжает работать. Как только условие перестает выполняться, цикл заканчивает свою работу.

Общий синтаксис

```
while <test>:              # Условное выражение test
      <statements1>        # Тело цикла
      if <test2>: break    # Выйти из цикла, пропустив часть else
      if <test3>: continue # Перейти в начало цикла, к выражению test1
else:
      <statements2>        # Выполняется, если выход из цикла
                           # производится не инструкцией break
```

In [ ]:

```
# Некоторое начальное условие
game_active = True

# Сам цикл while
while game_active:
    # Запускаем игру.
    # В какой-то момент времени необходимо изменить значение game_active на False.
    # Когда это случится, цикл закончит свою работу

# Действия по окончанию цикла
```

- Каждый цикл должен иметь начальное условие которое выполнятеся
- Ключевое слово `while` включает в себя проверяемое условие
- Код внутри цикла выполняется до тех пор, пока верно начальное условие
- Как только что-то меняется и начальное условие не выполняется, цикл перестает выполняться
- По окончании цикла выполняется код, написанный далее

Пример

В примере игра, которая продолжается до тех пор, пока у игрока есть силы

In [1]:

```
# Установим силы игрока - 5.
power = 5

# Играть можно пока сил не станет - 0.
while power > 0:
    print("You are still playing because your power is %d." % power)
    # игра начинается тут:
    #   возможная потеря сил
    # Допустим, с каждым разом сила у Люка уменьшается на 1
    power = power - 1

print("\nOh no, your power dropped to 0! Game Over.")
```

```
You are still playing because your power is 5.
You are still playing because your power is 4.
You are still playing because your power is 3.
You are still playing because your power is 2.
You are still playing because your power is 1.

Oh no, your power dropped to 0! Game Over.
```

Python 2.7

В Python 3 Вы всегда используете `input()`, и метод возвращает строку. В Python 2.7 `input()` - для ввода чисел, а для строк используйте `raw_input()`:

In [2]:

```
# Список содержит несколько имен
names = ['guido', 'tim', 'jesse']

# Просим пользователя ввести имя
new_name = raw_input("Please tell me someone I should know: ")

# Добавим новое имя в список
names.append(new_name)

# Напечатаем список, чтобы показать, что имя было добавлено
print(names)
```

```
Please tell me someone I should know: Jhon Doe
['guido', 'tim', 'jesse', 'Jhon Doe']
```

Использовать `input()` надо аккуратно, поскольку пользователь по сути может ввести что угодно.

Другое применение цикла while

Большинство программ работают до тех пор, пока вы не завершите работу, введя quit, exit ...etc. Рассмотрим пример организации цикла для подобного рода поведения:

In [3]:

```
# Создадим список
names = []

# Определим переменную new_name отличную от 'quit'.
new_name = ''

# Начнем цикл, который будет выполняться до тех пор, пока не набрано 'qui
t'.
while new_name != 'quit':
    # Попросим пользователя ввести имя
    new_name = input("Please tell me someone I should know or enter 'quit':
")

    # Добавим это имя в список который был создан ранее
    names.append(new_name)

# Покажем что все, что было набрано - добавлено в наш список.
print(names)
```

```
Please tell me someone I should know or enter 'quit': 'jessica'
Please tell me someone I should know or enter 'quit': 'diego'
Please tell me someone I should know or enter 'quit': 'quit'
['jessica', 'diego', 'quit']
```

Все хорошо, но! - 'quit' добавилось в лист. Давайте добавим `if` чтобы убрать этот баг:

In [10]:

```
#Ничего нового
names = []

new_name = ''

while new_name != 'quit':
    new_name = input("Please tell me someone I should know, or enter 'qui
t': ")
    if new_name != 'quit':
        names.append(new_name)

print(names)
```

```
Please tell me someone I should know, or enter 'quit': guido
Please tell me someone I should know, or enter 'quit': jesse
Please tell me someone I should know, or enter 'quit': jessica
Please tell me someone I should know, or enter 'quit': tim
Please tell me someone I should know, or enter 'quit': quit
['guido', 'jesse', 'jessica', 'tim']
```

Теперь у нас есть способ организации ввода для пользователя и контроля времени выполнения программы.

Использование цикла while для создания меню

In [33]:

```
# Give the user some context.
print("\nWelcome to the nature center. What would you like to do?")

# Set an initial value for choice other than the value for 'quit'.
choice = ''

# Start a loop that runs until the user enters the value for 'quit'.
while choice != 'q':
    # Give all the choices in a series of print statements.
    print("\n[1] Enter 1 to take a bicycle ride.")
    print("[2] Enter 2 to go for a run.")
    print("[3] Enter 3 to climb a mountain.")
    print("[q] Enter q to quit.")

    # Ask for the user's choice.
    choice = input("\nWhat would you like to do? ")

    # Respond to the user's choice.
    if choice == '1':
        print("\nHere's a bicycle. Have fun!\n")
    elif choice == '2':
        print("\nHere are some running shoes. Run fast!\n")
    elif choice == '3':
        print("\nHere's a map. Can you leave a trip plan for us?\n")
    elif choice == 'q':
        print("\nThanks for playing. See you later.\n")
    else:
        print("\nI don't understand that choice, please try again.\n")

# Print a message that we are all finished.
print("Thanks again, bye now.")
```

```
Welcome to the nature center. What would you like to do?

[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? 1

Here's a bicycle. Have fun!


[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? 3

Here's a map. Can you leave a trip plan for us?


[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? q

Thanks for playing. See you later.

Thanks again, bye now.
```

Ещё одно меню, но немного симпатичного кода

In [34]:

```
# Define the actions for each choice we want to offer.
def ride_bicycle():
    print("\nHere's a bicycle. Have fun!\n")

def go_running():
    print("\nHere are some running shoes. Run fast!\n")

def climb_mountain():
    print("\nHere's a map. Can you leave a trip plan for us?\n")

# Give the user some context.
print("\nWelcome to the nature center. What would you like to do?")

# Set an initial value for choice other than the value for 'quit'.
choice = ''

# Start a loop that runs until the user enters the value for 'quit'.
while choice != 'q':
    # Give all the choices in a series of print statements.
    print("\n[1] Enter 1 to take a bicycle ride.")
    print("[2] Enter 2 to go for a run.")
    print("[3] Enter 3 to climb a mountain.")
    print("[q] Enter q to quit.")

    # Ask for the user's choice.
    choice = input("\nWhat would you like to do? ")

    # Respond to the user's choice.
    if choice == '1':
        ride_bicycle()
    elif choice == '2':
        go_running()
    elif choice == '3':
        climb_mountain()
    elif choice == 'q':
        print("\nThanks for playing. See you later.\n")
    else:
        print("\nI don't understand that choice, please try again.\n")

# Print a message that we are all finished.
print("Thanks again, bye now.")
```

```
Welcome to the nature center. What would you like to do?

[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? 1

Here's a bicycle. Have fun!


[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? 3

Here's a map. Can you leave a trip plan for us?


[1] Enter 1 to take a bicycle ride.
[2] Enter 2 to go for a run.
[3] Enter 3 to climb a mountain.
[q] Enter q to quit.

What would you like to do? q

Thanks for playing. See you later.

Thanks again, bye now.
```

Использование while для обработки элементов списка

Мы можем достать элемент списка при помощи `pop()`. Цикл позволят нам пройти по элементам списка и применить к ним какие-то методы. Рассмотрим пример с обработкой неподтвержденных пользователей.

In [21]:

```
# Создадим два списка
unconfirmed_users = ['ada', 'billy', 'clarence', 'daria']
confirmed_users = []

# Пройдем по списку и проверим каждого пользователя
while len(unconfirmed_users) > 0:

    # Достанем последнего пользователя
    current_user = unconfirmed_users.pop()
    print("Confirming user %s...confirmed!" % current_user.title())

    # добавим его в лист подтвержденных
    confirmed_users.append(current_user)

# Пруфы обработок
print("\nUnconfirmed users:")
for user in unconfirmed_users:
    print('- ' + user.title())

print("\nConfirmed users:")
for user in confirmed_users:
    print('- ' + user.title())
```

```
Confirming user Daria...confirmed!
Confirming user Clarence...confirmed!
Confirming user Billy...confirmed!
Confirming user Ada...confirmed!

Unconfirmed users:

Confirmed users:
- Daria
- Clarence
- Billy
- Ada
```

Все прекрасно, но давайте сделаем небольшое улучшение. Сейчас мы обрабатываем только новых пользователей. Если пользователи появляются быстрее чем мы их обрабатываем, мы пропустим кучу народу, поэтому надо организовать взаимодействие вида 'первый пришел, первый вышел' (FIFO), поэтому каждый раз мы будет выбирать первого.

In [22]:

```
# Start with a list of unconfirmed users, and an empty list of confirmed us
ers.
unconfirmed_users = ['ada', 'billy', 'clarence', 'daria']
confirmed_users = []

# Work through the list, and confirm each user.
while len(unconfirmed_users) > 0:

    # Get the latest unconfirmed user, and process them.
    current_user = unconfirmed_users.pop(0)
    print("Confirming user %s...confirmed!" % current_user.title())

    # Move the current user to the list of confirmed users.
    confirmed_users.append(current_user)

# Prove that we have finished confirming all users.
print("\nUnconfirmed users:")
for user in unconfirmed_users:
    print('- ' + user.title())

print("\nConfirmed users:")
for user in confirmed_users:
    print('- ' + user.title())
```

```
Confirming user Ada...confirmed!
Confirming user Billy...confirmed!
Confirming user Clarence...confirmed!
Confirming user Daria...confirmed!

Unconfirmed users:

Confirmed users:
- Ada
- Billy
- Clarence
- Daria
```

Так намного лучше - мы обрабатываем пользователей в порядке их поступления!

Случайные бесконечные циклы

Иногда нужно выполнить цикл, но неизвестно, сколько итераций нужно совершить. Так часто возникают бесконечные циклы.

Давайте посмотрим пример:

In [ ]:

```
current_number = 1

# Count up to 5, printing the number each time.
while current_number <= 5:
    print(current_number)
```

In [ ]:

```
1
1
1
1
1
...
```

Не советую запускать код выше :) Как видите, цикл никогда не закончится, но пару советов, если дело плохо:

- В большинстве систем, Ctrl-C прервет выполнение работающей программы

Цикл бесконечный потому, что условие всегда выполняется. Для корректной работы необходимо в тело цикла добавить операцию инкремента переменной, значение которой проверяется в условии:

In [23]:

```
###highlight=[7]
current_number = 1

# Count up to 5, printing the number each time.
while current_number <= 5:
    print(current_number)
    current_number = current_number + 1
```

```
1
2
3
4
5
```

Если Вы не написали бесконечный цикл хотя бы один раз, значит Вы никогда не писали циклы. Если Вы все-таки решили узнать, как выглядит бесконечность - нажмите CTR + C и ищите логическую ошибку.

Ещё пара плохих примеров:

In [ ]:

```
current_number = 1

# Считаем до 5, печатая номер каждый раз.
while current_number <= 5:
    print(current_number)
    current_number = current_number - 1
```

In [ ]:

```
1
0
-1
-2
-3
...
```

Как видите, мы случайно начали считать не в ту сторону. Будьте внимательны!

Инструкции для работы с циклами

- **break** - Производит переход за пределы цикла
- **continue** - Производит переход в начало цикла
- **pass** - Ничего не делает, используется как заполнитель

Блок **else** - выполняется, если цикл завершился необычным образом (break не в счет)

Как это выглядит:

while :

```
<statements1>

if <test2>: break # Выйти из цикла, пропустив часть else

if <test3>: continue # Перейти в начало цикла, к выражению test1
```

else:

```
<statements2> # Выполняется, если не была использована инструкция 'b
reak'
```

Инструкции **break** и **continue** могут появляться в любом месте внутри тела цикла **while** (или *for*), но, как правило, они используются в условных инструкциях **if**, чтобы выполнить необходимое действие в ответ на некоторое условие.

Использование Continue

In [8]:

```
x = 10
while x:
    x = x - 1 # Или, x -= 1
    if x % 2 != 0: continue # Нечетное? — пропустить вывод
    print x,
```

```
8 6 4 2 0
```

Использование Break

In [ ]:

```
while 1:
    name = input('Enter name:')
    if name == 'stop': break
    age = input('Enter age: ')
    print('Hello', name, '=>', int(age) ** 2)
```

Использование Else

In [ ]:

```
x = y // 2 # Для значений y > 1
while x > 1:
    if y % x == 0: # Остаток
        print(y, 'has factor', x)
        break # Перешагнуть блок else
    x -= 1
else: # Нормальное завершение цикла
    print(y, 'is prime')
```

Цикл FOR

Цикл for в языке Python начинается со строки заголовка, где указывается переменная для присваивания (цель), а также объект, обход которого будет выполнен. Вслед за заголовком следует блок инструкций, которые требуется выполнить:

```
for <target> in <object>:   # Связывает элементы объекта с переменной цикла
    <statements> # Повторяющееся тело цикла: использует переменную цикла
    if <test>: break # Выход из цикла, минуя блок else
    if <test>: continue # Переход в начало цикла
else:
     <statements>   # Если не попали на инструкцию 'break'
```

Когда интерпретатор выполняет цикл for, он поочередно, один за другим, присваивает элементы объекта последовательности переменной цикла и выполняет тело цикла для каждого из них. Для обращения к текущему элементу последовательности в теле цикла обычно используется переменная цикла, как если бы это был курсор, шагающий от элемента к элементу. Имя (target), используемое в качестве переменной цикла (возможно, новой), которое указывается в заголовке цикла for, обычно находится в области видимости, где располагается сама инструкция for. Хотя она может быть изменена в теле цикла, тем не менее ей автоматически будет присвоен следующий элемент последовательности, когда управление вернется в начало цикла. После выхода из цикла эта переменная обычно все еще ссылается на последний элемент последовательности, если цикл не был завершен инструкцией break.

In [15]:

```
for x in ['spam', 'eggs', 'ham']:
    print x,
```

```
spam eggs ham
```