# Урок 05.02. Цикл for. Функция range

Вспомним ещё одну старую задачу:

text = ‘Python’

count = 0

while count < 5:

 print(text)

 count += 1

Давайте определим, что эта программа делает.

Данная программа выводи слово «Python» 5 раз.

Такую же программу можно написать с помощью цикла «for», но уже с новой функцией «range», переводится как «диапазон».

text = ‘Python’

for count in range(5):

 print(text)

Всё просто. Функция «range» создает объект, по которому можно перебирать как список чисел от 0 до 5. Но тут мы немного забежали вперед. И для того чтобы нам было проще понимать пока будем считать что range это список чисел от 0 до 5.

Такой ограниченный диапазон применения от 0 до n очень не удобен, и конечно же мы можем этот диапазон изменять.

А как мы его можем изменять, давайте посмотрим еще одну функцию help. И для этого в консоле («Python console») напишем команду «help(range)». В результате мы получим длинное описание этой функции. Нас интересуют следующие строчки:

Help on class range in module builtins:

class range(object)

 | range(stop) -> range object

 | range(start, stop[, step]) -> range object

 |

 | Return an object that produces a sequence of integers from start (inclusive)

 | to stop (exclusive) by step. range(i, j) produces i, i+1, i+2, ..., j-1.

 | start defaults to 0, and stop is omitted! range(4) produces 0, 1, 2, 3.

 | These are exactly the valid indices for a list of 4 elements.

 | When step is given, it specifies the increment (or decrement).

 |

 | Methods defined here:

 |

.........

 |

 | count(...)

 | rangeobject.count(value) -> integer -- return number of occurrences of value

 |

 | index(...)

 | rangeobject.index(value, [start, [stop]]) -> integer -- return index of value.

 | Raise ValueError if the value is not present.

 |

Из этого описания мы может узнать, что функция «range» может нам генерировать списки от start до stop с шагом step.

Рассмотрим такую задачу.

Алексей хочет поехать в кругосветное путешествие и для этого он каждый месяц откладывает в копилку определенное количество денег. И вот он хочет узнать сколько у него будет денег через определенное количество месяцев. Количество месяцев вводит пользователь.

total\_months = int(input(‘Сколько месяцев будем копить?’))

summ =

for month in range(total\_months):

 print(‘Месяц: ’, month)

 money = int(input(‘Сколько рублей откладываем? ’))

 summ += money

print(‘За’, total\_months, ’ты накопишь’, sum, ‘рублей’)

Запустим и убедимся, что программа работает.

Рассмотрим еще одну задачу.

У Саши интересный режим сна: он может проснуться когда угодно, хоть ночью, но засыпает всегда до того, как наступит полночь, обычно в 23 часа. А ещё он очень любит поесть. Он ест каждый час и если съедает больше 2000 калорий, то он просто падает спать.

Напишите программу, которая поможет Саше понять, всё ли с ним хорошо. Для этого нужно посчитать, сколько он всего съест калорий.

wake\_up = int(input("Во сколько проснулся? "))

calories\_sum = 0

for hour in range(wake\_up, 23):

 print("Сейчас", hour, "часов")

 calories = int(input("Сколько съел за час? "))

 calories\_sum += calories

 if calories\_sum > 2000:

 print("Хорошо поел, можно и поспать")

 break

print("Съедено калорий за день: ", calories\_sum)

## Типовые алгоритмы со счетными циклами

Мы всё больше и больше затрагиваем математике.

Представьте, у вас есть задача: сходить в магазин. И есть три дороги:

– долгий и безопасный (пешком и долго идти);

– быстрый и не безопасный (дворами и перебегаю дорогу где много машин);

– быстрый и безопасный (на трамвае), но нужно платить за проезд.

На самом деле могут быть еще пути: доехать на машине или заказать доставку еды на дом.

Так же и в программировании. И задача программиста это использовать наиболее эффективное решение. Для этого нужно учитывать скорость работы программы, объем потребляемой памяти, экономическую эффективность и простоту реализации.

Поэтому нужно знать различные алгоритмы решения типовых задач, которые очень часто будут встречаться на вашем пути в программировании.

Давайте рассмотрим задачу на стандартные алгоритмы.

Когда клиент отправляет в банк платежное поручение, он ставит под ним свою электронную подпись. У каждого платежного документа подпись своя и она не совпадает с другими документами. Без вычисления электронной подписи и для её проверки нельзя обойтись без простых чисел – такова магия криптографии. А простое число это такое число, которое делится только на 1 и на само себя. И именно это нам нужно решить.

number = int(input(‘Введите число: ’))

isPrime = True

for divider in range(2, number):

 if number % divider == 0:

 isPrime = False

 break

if isPrime:

 print(‘Число простое’)

else:

 print(‘Число составное’)

Попробуем… Давайте ещё раз попробуем 27644437 (оно простое и из википедии). Это число искалось долго.

В цифровых подписях используются гораздо большие числа в 10 раз длиннее.