Лекция 2

Циклы.

Строки.

Циклы: while

- Общий вид:

```
while condition:
    logic block
```

- Условие проверяется при входе в цикл и на каждой итерации

- Бесконечный цикл: `while True`

(используется вместе с *break*)

```
sum = 0
i = 0
while i <= 10:
    sum += i * i
    i += 1

print(sum)
```

```
> 385
```

Циклы: for

- Общий вид:
```
for i in iter:
    logic block
```

- Частный случай для *iterable*: **range(stop)** -> числа от 0 до **stop-1** (всего *stop* чисел).

- У *range* можно также менять начальное значение и шаг: **range(start, stop[, step]).** При этом значение *start* включается, а *stop* — нет.

Циклы: for

- Общий вид:

```
for i in iter:
    logic block
```

- Частный случай для *iterable*: **range(stop)** -> числа от 0 до **stop-1** (всего *stop* чисел).

- У *range* можно также менять начальное значение и шаг: **range(start, stop[, step]).** При этом значение *start* включается, а *stop* — нет.

```
s = 0
for i in range(10):
    s += i
print(s))

sum_odd_sq = 0
for i in range(1, 10, 2):
    sum_odd_sq += i*i
print(sum_odd_sq)
```

```
45
165
```

Циклы: for

- Что выведет такой код?

```
n = int(input())
mystery_sum = 0
for i in range(n, 0, -1):
    if mystery_sum % i == 0:
        mystery_sum += i
print(mystery_sum)
```

```
>10
???
```

Циклы: for

- Что выведет такой код?
- Значение *step* у функции *range* может быть отрицательным — тогда цикл проходит числа от *start* до *stop* в обратном порядке.

```
n = int(input())
mystery_sum = 0
for i in range(n, 0, -1):
    if mystery_sum % i == 0:
        mystery_sum += i
print(mystery_sum)
```

```
>10
21
```

```
10 + 5 + 3 + 2 + 1 = 21
```

Циклы: while vs for

- Часто условие в цикле *while* можно представить как проход по диапазону чисел — в этом случае использовать *for* проще и нагляднее.

```
sum_1 = 0
for i in range(11):
    sum_1 += i * i

####################

sum_2 = 0
i = 0
while i <= 10:
    sum_2 += i * i
    i += 1

print(sum_1, sum_2)
```

```
> 385 385
```

Циклы: while vs for

- Часто условие в цикле *while* можно представить как проход по диапазону чисел — в этом случае использовать *for* проще и нагляднее.

- Но изредка случается и наоборот :)

```
import math
n = int(input())
sum_1 = 0
b_len = int(math.log(n,2)) + 1
for i in range(b_len):
    sum_1 += n % 2
    n //= 2

##########################


n = int(input())
sum_2 = 0
while n > 0:
    sum_2 += n % 2
    n //= 2

print(sum_1, sum_2)
```

Циклы: break

- *break* позволяет немедленно выйти из текущего цикла
- Если находимся внутри вложенного цикла — выходим только из одного
- Удобно использовать при пользовательском вводе

Циклы: break

- *break* позволяет немедленно выйти из текущего цикла
- Если находимся внутри вложенного цикла — выходим только из одного
- Удобно использовать при пользовательском вводе

```
number = 23
guess = 0

while True:
    guess = int(input('Your guess:
'))
    if number == guess:
        print('Well done!')
        break
    elif number < guess:
        print('Too big.')
    else:
        print('Too small.')
```

```
>Your guess: 20
Too big.
>Your guess: 23
Well done!
```

Циклы: continue

- *continue* немедленно переходит к следующей итерации цикла (пропуская все инструкции после)

```
>Your guess: 1234
It is from 0 to 100.
>Your guess: 20
Too big.
>Your guess: 23
Well done!
2 trials made.
```

```
number = 23
cnt = 0

while True:
    guess = int(input('Your guess: '))
    if not (0 <= guess < 100):
        print('It is from 0 to 100.')
        continue
    if number == guess:
        print('Well done!')
        print(cnt + 1, 'trials made.')
        break
    elif number < guess:
        print('Too big.')
        cnt += 1
    else:
        print('Too small.')
        cnt += 1
```

Циклы: else

- После цикла можно поставить блок *else:* он будет выполняться после последнего прохода цикла (но не при прерывании цикла с помощью break)

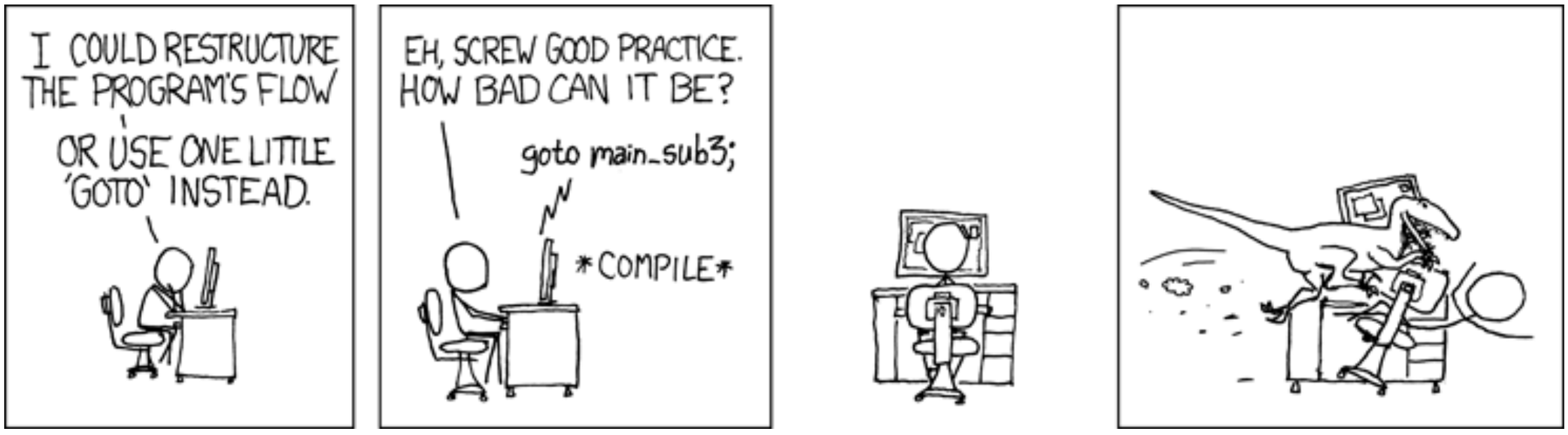
```
number = 23
guess = 0

for attempt in range(1, 8):
    guess = int(input('Your guess ({}/7): '.format(str(attempt))))
    if number == guess:
        print('Well done!')
        break
    elif number < guess:
        print('Too big.')
    else:
        print('Too small.')
else:
    print('You are out of attempts')
```

Циклы: советы

- Если заранее известно число итераций цикла, проще использовать **for**.

- Управляющие конструкции **break** и **continue** бывают полезны и необходимы. Но, как правило, лучше обойтись без них.



Строки

- Любая последовательность символов — строка (встроенный тип ***str***).

- Можно использовать русские буквы и вообще любые символы unicode.

- Двойные “” и одинарные ‘’ кавычки взаимозаменяемы.

- “\n”, “\’”, “\t” и т.д. — экранируемые символы

Строки

- Любая последовательность символов — строка (встроенный тип ***str***).
- Можно использовать русские буквы и вообще любые символы unicode.
- Двойные “” и одинарные ‘’ кавычки взаимозаменяемы.
- “\n”, “\’”, “\t” и т.д. — экранируемые символы

```
s1 = 'abc123_&'

s2 = 'Привет, мир'

s3 = "I'm hungry and you\'re not"


print(s1, '\n', s2, '\n', s3)
```

```
abc123_&
 Привет, мир
 I'm hungry and you're not
```

Строки

- Как избавиться от пробела в начале новой строки?

```
s1 = 'abc123_&'
s2 = 'Привет, мир'
s3 = "I'm hungry and you\'re not"

print(s1, '\n', s2, '\n', s3)

#print(???)
```

```
abc123_&
 Привет, мир
 I'm hungry and you're not
abc123_&
Привет, мир
I'm hungry and you're not
```

Строки

- Как избавиться от пробела в начале новой строки?

- Можно, например, конкатенировать строки (да, строки можно складывать).

- Но есть и более удобные способы!

```
s1 = 'abc123_&'
s2 = 'Привет, мир'
s3 = "I'm hungry and you\'re not"

print(s1, '\n', s2, '\n', s3)

print(s1 + '\n' + s2 + '\n' + s3)
```

```
abc123_&
 Привет, мир
 I'm hungry and you're not
abc123_&
Привет, мир
I'm hungry and you're not
```

Функция print

- Параметр ***sep*** по умолчанию равен одному пробелу, печатается между аргументами.

- Параметр ***end*** по умолчанию равен ‘\n’, печатается в конце.

- Форматирование строк: содержимое переменных вставляется вместо ‘***{}***’

- ***format*** — метод строки, его можно использовать и вне print

```
s1 = 'abc123_&'
s2 = 'Привет, мир'
s3 = "I'm hungry and you\'re not"

print(s1 + '\n' + s2 + '\n' + s3)

print(s1, s2, s3, sep='\n')

print(s1, s2, s3, sep='\t',
end='\n')

print('{}\t{}\t{}\n'.format(s1,
s2, s3))

print('{s1}\t{hello}\t{hungry}
\n'.format(s1=s1, hello=s2,
hungry=s3))
```

Строки: срезы

- К символам можно обращаться по индексам
- -1 — последний символ, -2 — предпоследний и т.д.

```
a = 'abcdef'
print(a[0], a[-1])
```

```
a f
```

Строки: срезы

- К символам можно обращаться по индексам
- -1 — последний символ, -2 — предпоследний и т.д.

- **Срезы (Slices)**: a[start:end], a[start:end:step] (точно так же, как у ***range***!)
- Однако, можно пропускать начальный/конечный индексы

- s[:] — копия исходной строки, s[::-1] — перевернутая строка.

```
a = 'abcdef'
print(a[0], a[-1])
print(a[1:5], a[1:-1])

print(a[2:], a[:2])
print(a[-2:], a[:-2])

print(a[::2], a[2::2])
print(a[::-1])
```

```
a f
bcde bcde
cdef ab
ef abcd
ace ce
fedcba
```

Строки: методы

Метод	Назначение
len(s)	Возвращает длину строки
s1 + s2	Конкатенация строк s1, s2
*s * n*	Повторение строки n раз
s.find(u)	Поиск 1-го вхождения *u* в строку *s*
s.count(u)	Подсчет числа вхождений u в строку s
s.upper(), s.lower()	Возвращает новую строку, все символы которой в верхнем/нижнем регистре
s.replace(x, y)	Заменяет все вхождения строки x на строку y
s.strip()	Обрезает пробельные символы в начале и конце строки
s.format()	Форматирование строки

Строки: методы

Примеры

```
e = '1,2,3'
print(e.find(','))
f = 'Time flies like a arrow; fruit flies
like a banana.'
f = f.replace('a ', 'an `, 1) # first only
print(f)
g = 'gooooooogle'
print(g.count('o'))
```

```
1
Time flies like an arrow; fruit flies
like an banana.
6
```

Встроенная документация

- **help()**: показывает краткую информацию о функциях и методах.

  Например: `help('').count`), `help(sum)`

- **dir()**: показывает список методов для данного объекта (переменной или типа). Методы, начинающиеся и оканчивающиеся на __ (нижнее подчеркивание) - специальные.

  Например: `dir(str)`, `dir(3)`

Полезные ссылки и книги

- https://docs.python.org/3/library/index.html - стандартная библиотека python 3 (eng). Здесь можно более подробно узнать про конкретную функцию или особенности синтаксиса.

- https://wiki.python.org/moin/TimeComplexity - таблица времени исполнения (асимптотического) стандартных операций со списками и другими структурами.

- Swaroop C.H. - A byte of python. - введение в python для не-программистов на русском языке.

- http://informatics.mccme.ru/course/view.php?id=156 - исходный курс Дениса Кириенко, ориентированный для школьников. На его основе был создан pythontutor.ru

Задачи

1. (Capitalize) Привести заданную строку *s* к регистру “первая буква большая, остальные — маленькие”.

2. (Lucky seven) На вход подается целое число *n*. Найти максимальное количество семерок, встречающихся в этом числе подряд.

Например, для 3774747 нужно вывести 2, а для 777 — 3.