

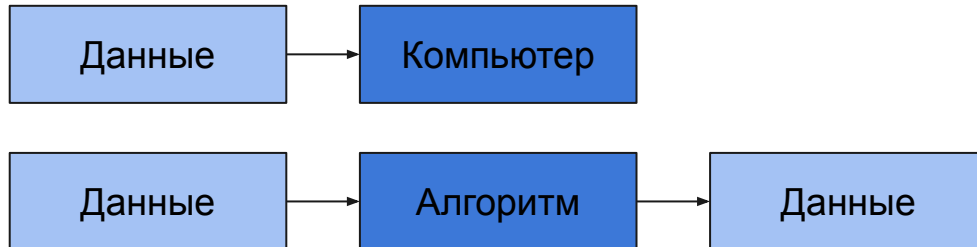
AaDSaPз_fZtH. Занятие 1

Часть 1. Введение в программирование

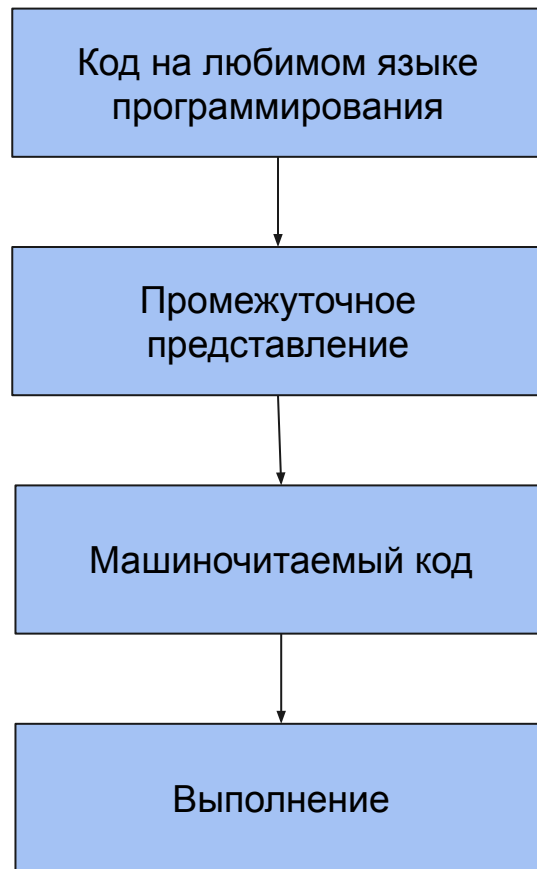
Емельянов Антон
login-const@mail.ru

Некоторые определения

- **Компьютерная программа** (она же приложение) — связка многочисленных строк специального текста.
- **Компьютерный код** — это специальный текст, состоящий из набора пошаговых инструкций.
- **Алгоритм** — это система последовательных операций (в соответствии с определёнными правилами) для решения конкретной задачи.
- **Язык программирования** — формальный язык, предназначенный для записи компьютерных программ.



Как выполняется программа / код?




Типы языков программирования

- **Компилируемый язык** — это такой язык, что программа, будучи скомпилированной, содержит инструкции целевой машины; этот машинный код непонятен обычным людям (только джедаям).
- **Интерпретируемый язык** — это такой, в котором инструкции не исполняются целевой машиной, а считываются и исполняются другой программой (которая обычно написана на языке целевой машины).

```
3 | Disassembly of section .text:
4 |
5 | 0000000000000000 :
6 | 0: 55 push rbp
7 | 1: 48 89 e5 mov rbp, rsp
8 | 4: 89 7d fc mov DWORD PTR [rbp-0x4], edi
9 | 7: 89 75 f8 mov DWORD PTR [rbp-0x8], esi
10 | a: 8b 75 fc mov esi, DWORD PTR [rbp-0x4]
11 | d: 0f af 75 f8 imul esi, DWORD PTR [rbp-0x8]
12 | 11: 89 f0 mov eax, esi
13 | 13: 5d pop rbp
14 | 14: c3 ret
```





АаDSaPз_fZtH. Занятие 1

Часть 2. Введение в Python

Емельянов Антон
login-const@mail.ru

- **Python** в русском языке распространено название *питон*) — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.



- Создан в 1991 году
- Автор Гвидо ван Россум
- Простой в использовании
- Свободный и имеет открытый исходный код
- Высокоуровневый
- Имеет динамическую типизацию

- Интерпретируемый

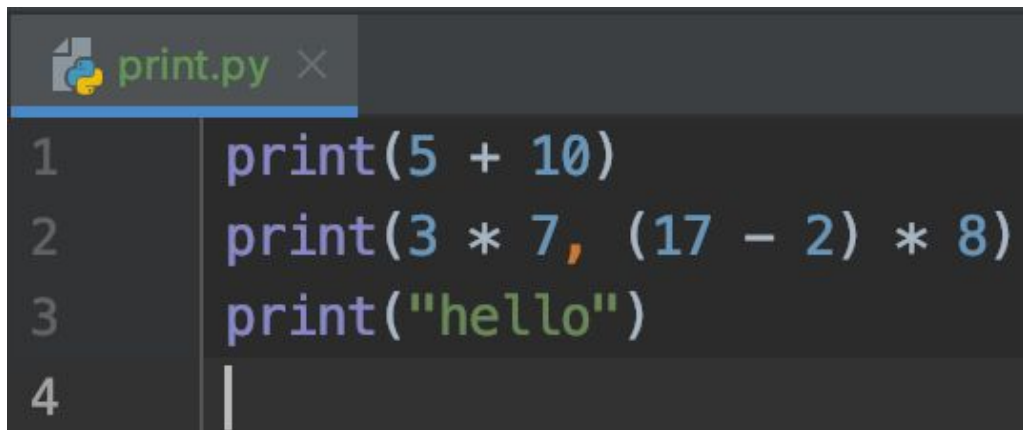
```
(base) CAB-WSM-0003050:L1.intro-python 4169895296 python3 -m dis hello.py
1          0 LOAD_NAME           0 (print)
          2 LOAD_CONST          0 ('hello')
          4 CALL_FUNCTION         1
          6 POP_TOP
          8 LOAD_CONST          1 (None)
         10 RETURN_VALUE
```


- Объектно-ориентированный — в питоне все является объектом

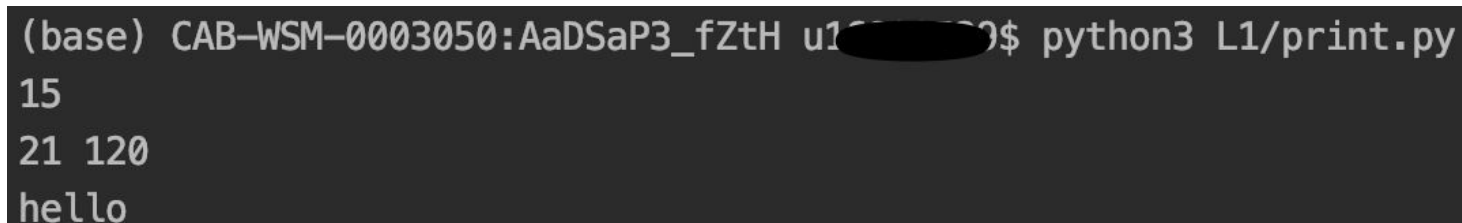
```
[>>> dir(print)
['__call__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__self__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__text_signature__']
>>>
```

Функция print

- Для печати значений в Питоне есть функция `print()`. Внутри круглых скобок через запятую мы пишем то, что хотим вывести.



```
print.py ×  
1 print(5 + 10)  
2 print(3 * 7, (17 - 2) * 8)  
3 print("hello")  
4 |
```



```
(base) CAB-WSM-0003050:AaDSaP3_fZtH u1[REDACTED]$ python3 L1/print.py  
15  
21 120  
hello
```

Функция input

- Для ввода данных в программу мы используем функцию `input()`. Она считывает одну строку.

```
sum1.py x
1 a = input()
2 b = input()
3 s = a + b
4 print(s)
5 |
```

```
5
7
57
```

Основные типы данных и операции

- **Тип int (long)** — целые числа не имеют ограничений на значения (All integers are implemented as “long” integer objects of arbitrary size. URL: <https://docs.python.org/3.6/c-api/long.html>).

```
int.py ×  
1 a = int(input())  
2 b = 10 ** 100  
3  
4 print(a + b)  
5
```

[illegible]

Основные типы данных и операции

- Тип `float` — вещественные числа (числа с плавающей запятой). Максимальное число может зависеть от платформы.
 - `-inf` и `inf` это то что это числа обозначающие плюс минус бесконечность.

```
float.py x
1 a = float(input())
2 b = 1.7976931348623157e+308
3 x = 1.7 * 10e307 * 1.7 * 10e307
4 y = -10e9999
5
6 print(a + b)
7 print(x, y, x + y)
8 print(10.0 ** 309)
9
```

```
1
1.7976931348623157e+308
inf -inf nan
Traceback (most recent call last):
  File "L1/float.py", line 8, in <module>
    print(10.0 ** 309)
OverflowError: (34, 'Result too large')
```

Основные типы данных и операции

- Тип `str` — базовый тип представляющий из себя *неизменяемую* последовательность символов; `str` от «string» — «строка».

```
str.py x
1  s1 = input()
2  s2 = "world"
3  s3 = s1 + " " + s2
4  s4 = str(10)
5
6  print(s3, s3[1], 'Кот' 'обус', sep="\t")
7
8  # Одиночные кавычки. Часто встречаемый вариант записи.
9  my_str = 'а внутри "можно" поместить обычные'
10 # Кавычки.
11 my_str = "а внутри 'можно' поместить одиночные"
12 # Три одиночных кавычки. Удобно для записей в несколько строк
13 my_str = '''В трёх одиночных
14           кавычках'''
15 # Тройные кавычки. Общепринятый способ для строк документации.
16 my_str = """Three double quotes"""
```

```
hi
hi world      i      Котобус
```

Основные типы данных и операции

- Тип `bool` — представлен двумя постоянными значениями `False` и `True`. Значения используются для представления истинности.

```
bool.py x
1 my_bool_true = True
2 print(my_bool_true, bool(10), bool(-10), bool('some'))
3
4 my_bool_false = False
5 print(my_bool_false, bool(0), bool(''), bool())
6
```

```
True True True True
False False False False
```

Основные типы данных и операции

- Арифметические операции: + - * ** % //
- Бинарные операции:
 - & битовое И (AND)
 - | битовое ИЛИ (OR)
 - ^ битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)
 - ~ битовое ОТРИЦАНИЕ (NOT) — унарная операция

operation1.py		
1	<code>print(10 / 3 * (3 % 5) // 3 + 1)</code>	4.0
2	<code>print(7 & 3)</code>	3
3	<code>print(7 and 3)</code>	3
4	<code>print(7 3)</code>	7
5	<code>print(7 or 3)</code>	7

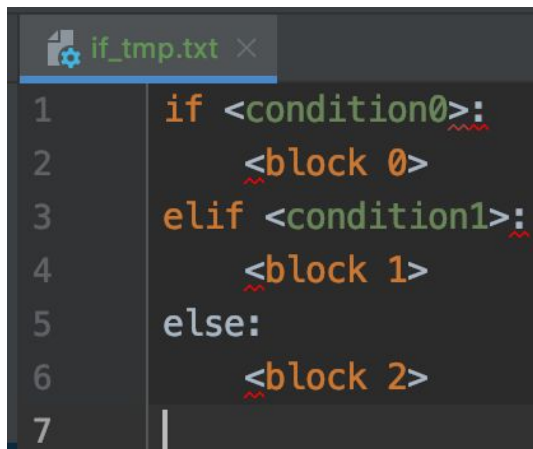
Основные типы данных и операции

- Логические операции:
 - < > <= >= == !=
 - not and or

operation2.py		
1	<code>print(25 <= 7 and 0)</code>	False
2	<code>print(25 < 7 or 1)</code>	1
3	<code>print(10 or False)</code>	10
4	<code>print(not False)</code>	True
5	<code>print(bool(10) == bool(-10))</code>	True

Условные конструкции

- Условная инструкция в Питоне имеет следующий синтаксис:

A screenshot of a code editor window titled 'if_tmp.txt'. The editor shows a Python conditional statement with line numbers 1 through 7 on the left. The code is: 1 if <condition0>:, 2 <block 0>, 3 elif <condition1>:, 4 <block 1>, 5 else:, 6 <block 2>, 7 |. The keywords 'if', 'elif', and 'else' are in green, and the placeholders for conditions and blocks are in orange. Red squiggly lines are under the colons and the 'elif' keyword.

```
1  if <condition0>:
2      <block 0>
3  elif <condition1>:
4      <block 1>
5  else:
6      <block 2>
7  |
```

- block 0 будет выполнен, если condition0 истинно. Если Условие ложно, будет выполнен block 1, если condition1 истинно. Иначе будет выполнен block 2.
- В условной инструкции может отсутствовать слово **else** и последующий блок. Такая инструкция называется неполным ветвлением. Также может отсутствовать **elif**.

Вложенные условные конструкции

- Внутри условных инструкций можно использовать любые инструкции языка Питон, в том числе и условную инструкцию. Получаем вложенное ветвление - после одной развилки в ходе исполнения программы появляется другая развилка. При этом вложенные блоки имеют больший размер отступа (например, 8 пробелов).

```
if1.py x
1  x = int(input())
2  y = int(input())
3  if x > 0:
4      if y > 0:                # x > 0, y > 0
5          print("Первая четверть")
6      else:                   # x > 0, y < 0
7          print("Четвертая четверть")
8  else:
9      if y > 0:                # x < 0, y > 0
10         print("Вторая четверть")
11     else:                   # x < 0, y < 0
12         print("Третья четверть")
13
```

Каскадные условные конструкции

- Пример программы, определяющий четверть координатной плоскости, можно переписать используя “каскадную” последовательность операций **if... elif... else:**

```
if2.py x
1  x = int(input())
2  y = int(input())
3  if x > 0 and y > 0:
4      print("Первая четверть")
5  elif x > 0 and y < 0:
6      print("Четвертая четверть")
7  elif y > 0:
8      print("Вторая четверть")
9  else:
10     print("Третья четверть")
11
```

Цикл for

- Цикл **for**, также называемый циклом с параметром, в языке Питон богат возможностями. В цикле **for** указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном.

```
for1.py x
1     idx = 1
2     for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
3         print('#', idx, ' color of rainbow is ', color, sep='')
4         idx += 1
5
```

```
#1 color of rainbow is red
#2 color of rainbow is orange
#3 color of rainbow is yellow
```

Функция range

- Как правило, циклы **for** используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.
- Для повторения цикла некоторое заданное число раз **n** можно использовать цикл **for** вместе с функцией **range**.
- Есть три способа вызова **range()**:
 1. **range(стоп)** берет один аргумент
 2. **range(старт, стоп)** берет два аргумента
 3. **range(старт, стоп, шаг)** берет три аргумента

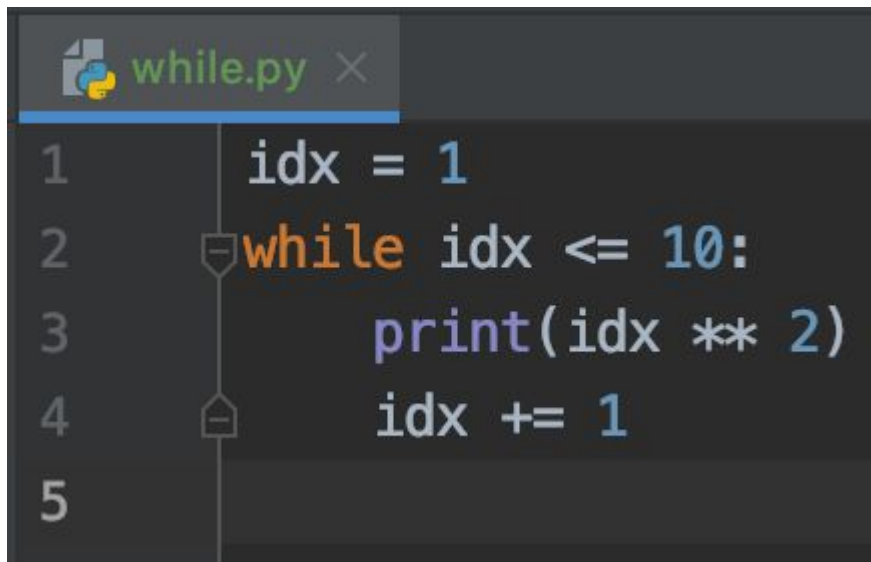
Функция range

```
range1.py ×  
1  for i in range(3, 16, 3):  
2      quotient = i / 3  
3      print(i, "делится на 3, результат " + int(quotient) + ".")  
4
```

```
3 делится на 3, результат 1.  
6 делится на 3, результат 2.  
9 делится на 3, результат 3.  
12 делится на 3, результат 4.  
15 делится на 3, результат 5.
```

Цикл while

- Цикл **while** (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл **while** используется, когда невозможно определить точное значение количества проходов исполнения цикла.

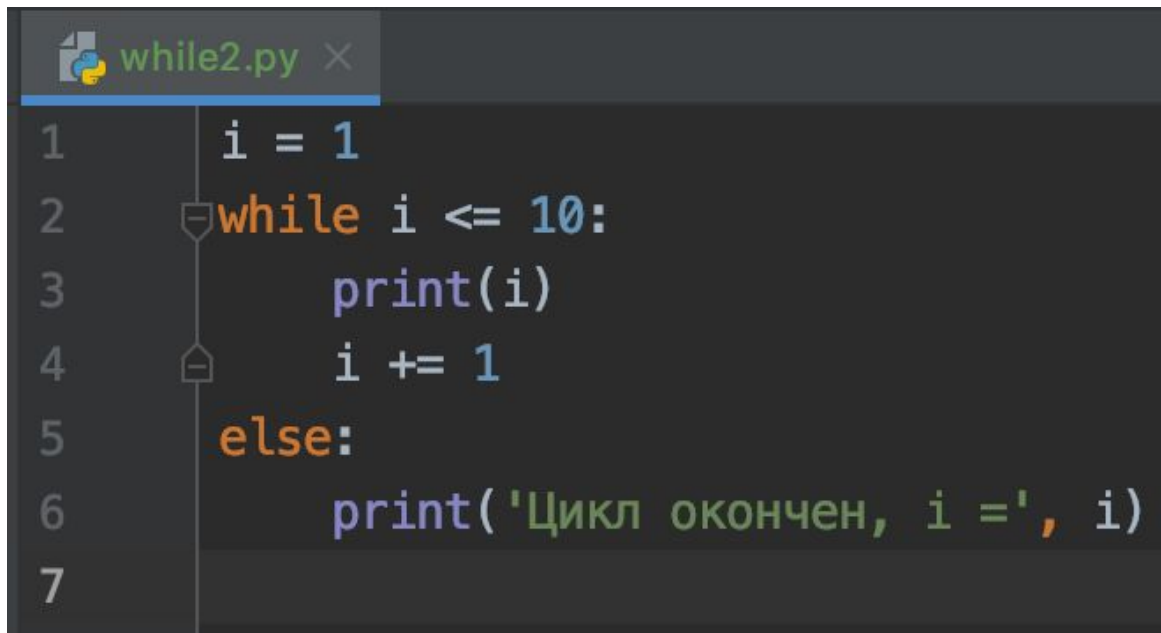


```
while.py x
1  idx = 1
2  while idx <= 10:
3      print(idx ** 2)
4      idx += 1
5
```

```
1
4
9
16
25
36
49
64
81
100
```


Цикл while

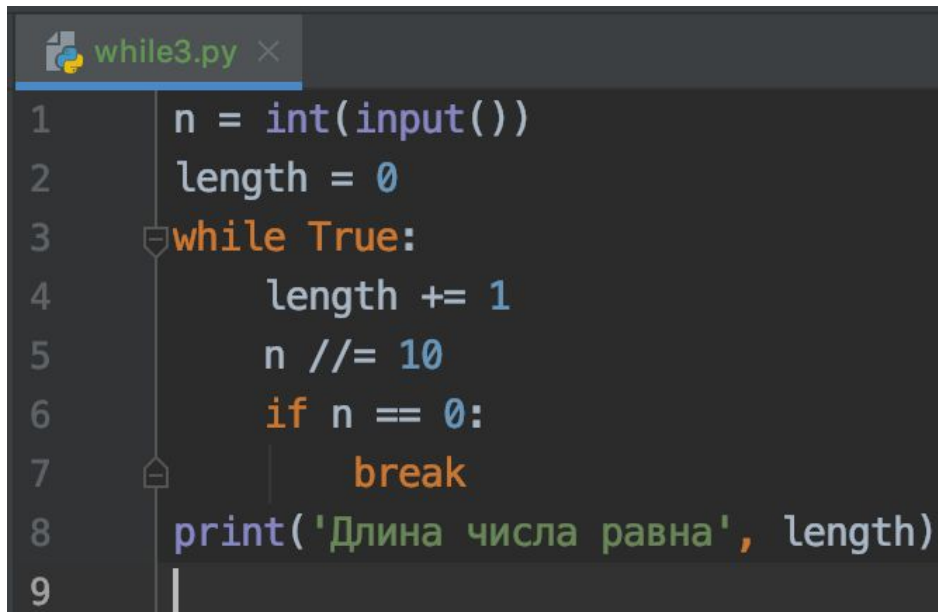
- После тела цикла можно написать слово **else:** и после него блок операций, который будет выполнен *один раз* после окончания цикла, когда проверяемое условие станет неверно:



```
while2.py x
1 i = 1
2 while i <= 10:
3     print(i)
4     i += 1
5 else:
6     print('Цикл окончен, i =', i)
7
```

Цикл while

- Смысл **else** появляется только вместе с инструкцией **break**. Если во время выполнения Питон встречает инструкцию **break** внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка **else** исполняться не будет. Разумеется, инструкцию **break** осмысленно вызывать только внутри инструкции **if**, то есть она должна выполняться только при выполнении какого-то особенного условия.



```
while3.py ×
1  n = int(input())
2  length = 0
3  while True:
4      length += 1
5      n //= 10
6      if n == 0:
7          break
8  print('Длина числа равна', length)
9
```

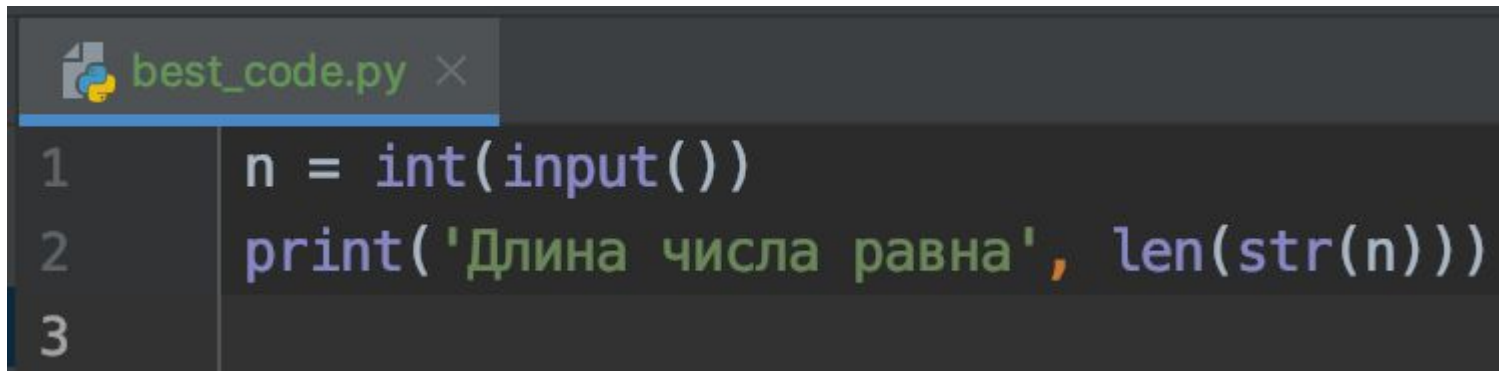
Цикл while

- Хороший код:

```
while4.py x
1  n = int(input())
2  length = 0
3  while n != 0:
4      length += 1
5      n //= 10
6  print('Длина числа равна', length)
7  |
```

Правильный код

- На питоне можно предложить и более изящное решение вычисления длины числа:



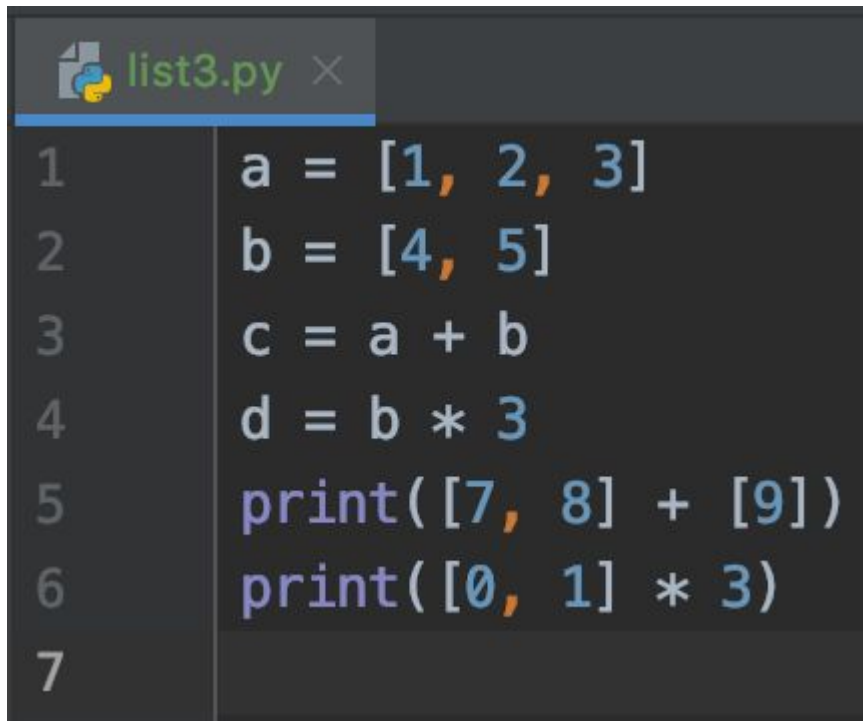
```
best_code.py x
1 n = int(input())
2 print('Длина числа равна', len(str(n)))
3
```

- Список представляет собой последовательность элементов, пронумерованных от 0, как символы в строке. Список можно задать перечислением элементов списка в квадратных скобках.

```
list1.py x
1  rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']
2  print(rainbow[0])
3  rainbow[0] = 'красный'
4  print('Выведем радугу')
5  for i in range(len(rainbow)):
6      print(rainbow[i])
7  |
```

- Как прочитать список?

```
list2.py x
1  a = [] # заводим пустой список
2  for i in range(int(input())): # считываем количество элемент в списке
3      new_element = int(input()) # считываем очередной элемент
4      a.append(new_element)      # добавляем его в список
5      # последние две строки можно было заменить одной:
6      # a.append(int(input()))
7  print(a)
8  |
```



```
1 a = [1, 2, 3]
2 b = [4, 5]
3 c = a + b
4 d = b * 3
5 print([7, 8] + [9])
6 print([0, 1] * 3)
7
```

Индексирование и срезы

- Работает со списками, кортежами и строками

```
In [194]: lst = list(range(20))
```

```
In [195]: lst[1]
```

```
Out[195]: 1
```

```
In [196]: lst[-1]
```

```
Out[196]: 19
```

```
In [197]: lst[-4]
```

```
Out[197]: 16
```

```
In [198]: lst[1:5]
```

```
Out[198]: [1, 2, 3, 4]
```

```
In [199]: lst[1:14:2]
```

```
Out[199]: [1, 3, 5, 7, 9, 11, 13]
```

```
In [200]: lst[-1:1:-2]
```

```
Out[200]: [19, 17, 15, 13, 11, 9, 7, 5, 3]
```

```
In [201]: lst[::-2]
```

```
Out[201]: [19, 17, 15, 13, 11, 9, 7, 5, 3, 1]
```


```
In [202]: lst[:]
```

```
Out[202]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```


Функции списка

```
list4.py x
1  lst = list(range(4))
2  print(lst)
3  # Вставка в конец
4  lst.append(4)
5  print(lst)
6  # Вставка на позицию
7  lst.insert(2, -1)
8  print(lst)
9  # Выталкивание
10 last = lst.pop(-1)
11 print(lst, last)
12 # Удаление
13 lst.remove(3)
14 print(lst)
15
```

```
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
[0, 1, -1, 2, 3, 4]
[0, 1, -1, 2, 3] 4
[0, 1, -1, 2]
```



АаDSaPз_fZtH. Занятие 1

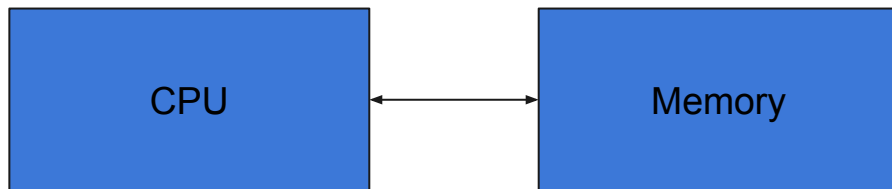
Часть 3. Сложность вычислений.

Организация списка в Python

Емельянов Антон
login-const@mail.ru

Сложность вычислений

- Ресурсы:
 - Память (M)
 - Процессорное время (T)



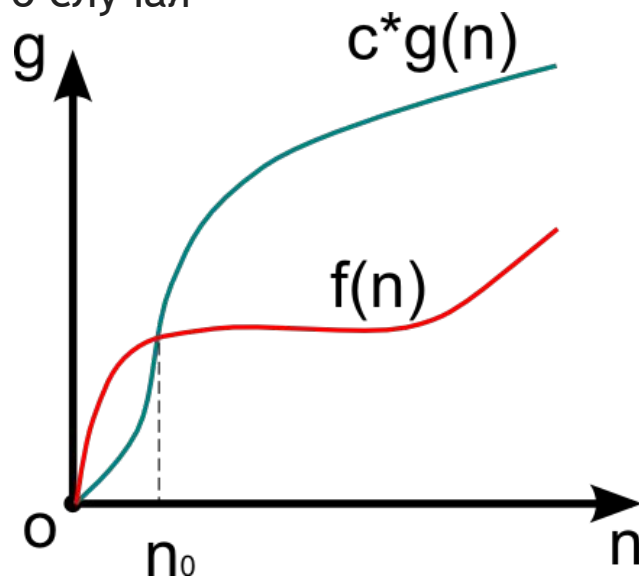
- Память vs Время — tradeoff
- Оценка сложности алгоритма



- Размер входа: N
- Время работы алгоритма: $T(N)$

Сложность вычислений

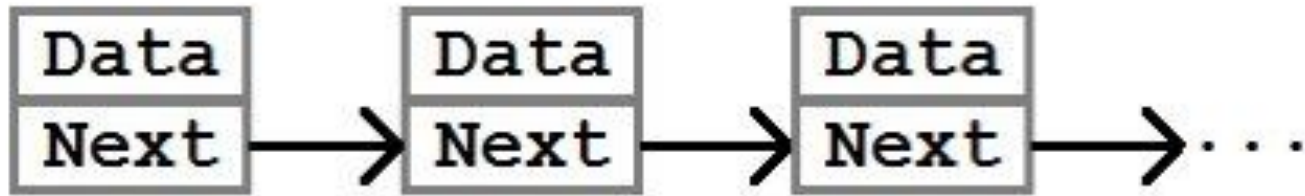
- O - оценка для худшего случая



- Функция $g(n)$ в данном случае асимптотически-точная оценка $f(n)$. Если $f(n)$ - функция сложности алгоритма, то порядок сложности определяется как $f(n) - O(g(n))$. Данное выражение определяет класс функций, которые растут не быстрее, чем $g(n)$ с точностью до константного множителя.

Устройство списка

- **Связный список** — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка.



Список в python. Память

- Обычно список занимает в памяти “больше”, чем число элементов

Элементы списка					
Память					

Добавление элемента в конец

- Операция append в python

Элементы списка					
Память					

Добавление элемента в конец

- Операция append в python

Элементы списка							
Память							

Добавление элемента в конец

- Операция append в python
- Сложность $O(1)$

Элементы списка							
Память							

Добавление элемента в середину

- Операция insert в python

Элементы списка	1	2	3	4	5		
Память							

Элементы списка	1	2	-1	3	4	5	
Память							

Добавление элемента в середину

- Операция insert в python
- Сложность $O(N)$

Элементы списка	1	2	3	4	5		
Память							

Элементы списка	1	2	-1	3	4	5	
Память							

Домашнее задание

- Прорешать задачи тут
http://pythontutor.ru/lessons/inout_and_arithmetic_operations/
 - Пункты: 1, 2, 4, 6, 7
- Придумать алгоритм функций remove и pop для списков

- <https://tproger.ru/translations/learn-to-write-some-simple-code/>
- <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC>
- https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F
- <https://tproger.ru/translations/programming-concepts-compilation-vs-interpretation/>
- <https://ru.wikipedia.org/wiki/Python>
- <https://pyprog.pro/python/py/nums/float.html>
- <https://pythonz.net/references/named/str/>
- <https://pythonz.net/references/named/int/>
- <https://pythonz.net/references/named/bool/>
- <https://python-scripts.com/range>
- <https://habr.com/ru/post/273045/>
- <https://tproger.ru/translations/linked-list-for-beginners/>
- https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA
- <https://e-maxx.ru/bookz/files/cormen.pdf>