

# Основы и методология программирования

## Семинар 2

Логический тип данных

Ветвление

Простые циклы

# Битовые, логические операции, тип bool

```
a = 5 # 0b101
b = 6 # 0b110
```

```
c = a & b # 0b100 == 4
d = a | b # 0b111 == 7
e = a ^ b # 0b011 == 3
f = ~ a # 0b1...11111010 == -6
```

```
print (bool(a >> 3)) # сдвиг вправо (деление на 2^k)
print (bool(a << 3)) # сдвиг влево (умножение на 2^k)
print (bool(a) != bool(b)) # xor
```

not	Логическое НЕ	x = True; not x - False.
and	Логическое И	x = False; y = True; x and y возвращает False, поскольку x равно False. В этом случае Python не станет проверять значение y, так как уже знает, что левая часть выражения 'and' равняется False
or	Логическое ИЛИ	x = True; y = False; x or y даёт True. Тоже "ленивый".

# Условный оператор

1. if
2. if-else
3. if-elif-else

```
# 1
if a > 0:
    b = 10
    c = 20
# 1a
if (not a): b = 0
# 2
if a > b > c:
    b = 10
    c = 20
else:
    b = 20
    c = 10
```

```
# 3
if ~a and ~b:
    c = 10
elif (a > 10) and (b > 0):
    a = 10
else:
    a = 0

# 3a
if ~a and ~b:
    c = 10
else:
    if (a > 10) and (b > 0):
        a = 10
    else:
        a = 0
        print ("Внутри else")
print ("Вне блоков")
```

**Блок операторов выделяется отступом в 4 пробела от предыдущего уровня (!)**

Официальный Style Guide целиком:  
<https://www.python.org/dev/peps/pep-0008/>

# Проверка входных данных и преобразование типов переменных

1. Проверка не требуется (данные гарантированно? хорошие)

```
n = int(input("Введите Ваш возраст: "))
```

1. Проверка целых неотрицательных

```
if (str.isdigit()):  
    n = int(str)
```

1. Проверка целых

```
a = "-8".lstrip("-").isdigit()
```

1. Более сложные проверки (с заданием формата входных данных)

Регулярные выражения (пока рано голову забивать, пример, который со временем будет понятен как 2\*2)

```
_float_regexp = re.compile(r"^[+-]?(?:\b[0-9]+(?:\.[0-9]*)?|\.[0-9]+\b)(?:[eE][+-]?[0-9]+\b)?$").match
```

```
def is_float_re(str):  
    return True if _float_regexp(str) else False
```

`lstrip([chars])` Возвращает копию указанной строки, с начала (слева | — left, есть **strip** и **rstrip**) которой устранены указанные символы.

# Задача 1

Написать метод для вычисления приближенного значения  $n$ -го члена ряда Фибоначчи по формуле Бине:

$$U_n = \frac{b^n - (-b^{-n})}{2b - 1}$$
$$b = \frac{1 + \sqrt{5}}{2}$$

В основной программе, вводя значения  $n$ , вычислять и выводить приближенное вещественное значение  $n$ -го члена ряда Фибоначчи. Окончание работы программы – ввод нулевого или отрицательного значения  $n$ .

# Задача 1

```
import math

n = 0          # Номер члена ряда
b = 0.0        # Вспомогательная переменная
un = 0.0       # Оценка по формуле Бине
res = 0        # Целочисленное значение члена
line = ""      # Строка для хранения пользовательских данных

line = input("Введите член ряда: ")
if (line.isdigit()):
    n = int(line)
else:
    exit(1)     # Код возврата в ОС
b = (1 + math.sqrt(5)) / 2
un = (math.pow(b, n) - math.pow(-b, -n)) / (2 * b - 1)
res = int(un + 0.5)
print("Член ряда #%d равен %d" % (n, res))
```

# Задача 2

Вводится радиус, вычислить  $S$  круга и периметр окружности.

```
import math

rad_str = input("Введите радиус: ")
if (rad_str.isdigit()):
    rad = int(rad_str)
else:
    exit(1)
print("P окружности равен ", 2 * math.pi * rad, sep='')
print("S круга равна", rad * math.pi**2 / 2) # у возведения в степень приоритет
ВЫШЕ
```

# Цикл while

```
count = 0
while (count < 9):          # условие продолжения
    print ('The count is:', count)  # тело цикла
    count = count + 1        # всё ещё тело цикла
```

Ключевое слово	Описание
<b>break</b>	Прерывает выполнение цикла
<b>continue</b>	Переходит на следующую итерацию
<b>pass</b>	Заменитель тела цикла, когда требуется тело цикла, но ничего делать не нужно



# Задача 3

Пользователь вводит число. Посчитать квадратный корень из него по методу Ньютона с точностью 0,1. Нельзя пользоваться стандартной функцией sqrt().

Метод Ньютона:

1. Заданы число (A), начальное приближение( $x_k$ ) и требуемая точность( $\epsilon$ ).
2. Следующее приближение вычисляется по формуле:

$$x_{k+1} = \frac{1}{2} \left( x_k + \frac{A}{x_k} \right)$$

Если на  $k$ -том шаге точность достигнута, то ответ найден, иначе - повторить с новым начальным приближением

# Задача 3

```
import math

eps = 0.1 # задаем точность вычислений
a = 0     # число из которого будет извлекаться корень
xk = xk1 = 0 #промежуточные результаты

while True:
    # ввод и проверка условий
    str_a = input("Введите число, для выхода введите '0':")
    if str_a.startswith('-'): # если начинается со знака минус
        print("Нельзя извлекать корень из отрицательного числа")
        continue # повторяем запрос ввода
    if not str_a.isdigit():
        print("Нужно ввести положительное целое число!")
        continue
    a = int(str_a)
    if not a : break #если ввели 0 - выходим
    # основной цикл
    xk1 = a / 2 # первое приближение пусть будет половина от a, можно взять любое
    while (math.fabs(xk1 - xk) > eps): #пока еще точность не достигнута
        xk = xk1
        xk1 = 0.5 * (xk + a / xk)
    print ("Квадратный корень из числа %d равен %g с точностью %g" % (a, round(xk1, 2), eps))
```

# Условная операция (тернарная операция)

`X = A if <условие> else B`

```
# Написать программу, так обменивающий значения трех переменных x, y, z,  
# чтобы выполнялось требование: x < y < z.
```

```
x = 10  
y = 7  
z = 3
```

```
a1 = a2 = a3 = 0 # временные переменные
```

```
a1 = z if (y > z and x > z) else y if (x > y) else x  
a2 = y # a2 - самостоятельно :)  
a3 = z if (y < z and x < z) else y if (y > x) else x  
print ("%d < %d < %d" % (a1, a2, a3))
```