Python.

Функции (начало).

Как мы уже видели, в языке Python есть стандартные функции, например, round(), int(), float(). Но также, как и в других языках, имеется возможность создавать собственные функции.

Общий вид описания собственной функции следующий

```
def имя_функции(параметры):
    оператор1
    оператор2

    ...
    операторN
    return результат
```

После ключевого слова def следует имя функции, обязательно с круглыми скобками, в которых указываются параметры. Если параметров нет, то скобки все равно обязательно должны быть. После скобок ставится двоеточие. Тело функции пишется со следующей строки, обязательно с отступами. Результат, который должна возвратить функция, записывается после ключевого слова `return`.

Напишем функцию, которая возвращает 1, если на вход подано положительное число и 0 в противном случае.

```
def is_p(x):
    if x > 0:
        return 1
    else:
        return 0


a = int(input())
print(is_p(a))
```

Обратите внимание, что может быть несколько операторов `return`. Описание функции стоит отделять двумя пустыми строчками от основной программы и от описания других функций.

В языке Python в отличии от языка Паскаль, нет процедур. Однако есть возможность делать функции, которые ничего не возвращают и вызываются как процедуры. Напишем функцию, которая будет печатать слово «Привет!», если параметр равен 1 и «Пока!» при любом другом значении.

```
def priv(x):
    if x == 1:
        print("Привет!")
    else:
        print("Пока!")


priv(1)
priv(7)
```

В результате работы данной программы будет напечатано

```
Привет!
Пока!
```

Все переменные в функциях локальны. Однако, если некоторой переменой задать значение до описания функции, она будет видна внутри функции.

```
z = 20


def summ(x, y):
    return x + y + z


print(summ(1,2))
print(z)
```

При выполнении этой программы будет напечатано 23 и 20.

Изменим программу

```
z = 20


def summ(x, y):
    z = 30
    return x + y + z


print(summ(1,2))
print(z)
```

При выполнении этой программы будет напечатано 33 и 20. Почему?

Дело в том, что при выполнении оператора `z = 30` создастся новая переменная `z`, которая будет локальна внутри функции. Та же переменная `z`, которая было создана до функции, останется неизменной.

Функции могут возвращать не одно, а несколько значений.

```
def minmax(x, y):
    if x > y:
        return y, x
    else:
        return x, y


a = int(input())
b = int(input())
a, b = minmax(a, b)
print(a, b)
```

В результате работы программы наименьшее число окажется в переменной `a` , а наибольшее – в переменной `b`.

Вещественные числа.

Естественно, что кроме целых чисел в языке Python используются и вещественные числа. Для приведения к вещественному типу используется функция float().

```
a = float(input())
print(2 * a)
```

Дробная часть отделяется от целой с помощью точки.

Бывает необходимо привести дробное число к целому. Имеется несколько способов выполнить эту задачу. Конечно, это можно сделать с помощью функции преобразования к целому числу ***int()***. В этом случае отбрасывается дробная часть.

```
print(int(2.3214))
print(int(5.987))
print(int(-2.3214))
print(int(-5.879))
```

При выполнении этой программы будут выведены числа

```
2
5
-2
-5
```

Для округления числа до ближайшего целого используется функция `round()` с одним параметром

```
print(round(2.3214))
print(round(5.987))
print(round(-2.3214))
print(round(-5.879))
```

В этом случае получим

```
2
6
-2
-6
```

Однако при выполнении следующей программы

```
print(round(4.5))
print(round(5.5))
print(round(-4.5))
print(round(-5.5))
```

получим

```
4
6
-4
-6
```

Дело в том, что при дробной части равной ***0.5*** округление происходит до ближайшего ***четного*** числа.

Если использовать функцию `round()` с двумя параметрами, то второй параметр будет указывать до какого знака после запятой будет округлено число.

```
print(round(1.23456789,5))
print(round(1.23456789,4))
print(round(1.23456789,2))
print(round(1.23456789,1))
```

будет напечатано

```
1.23457
1.2346
1.23
1.2
```

Если второй параметр сделать отрицательным, то число будет округляться до десятков, сотен, тысяч и т. д.

```
print(round(1234567.89,-1))
print(round(1234567.89,-2))
print(round(1234567.89,-4))
print(round(1234567.89,-5))
```

получим

```
1234570.0
1234600.0
1230000.0
1200000.0
```

Если нет необходимости округлять числа, а надо только ограничить количество знаков при выводе, можно воспользоваться функцией format().

В общем виде синтаксис записи этой функции выглядит следующим образом

`format(value, format_spec)`

где

- **`value`** - форматируемое значение,
- **`format_spec`** - спецификации формата

Есть много вариантов спецификации, однако мы пока рассмотрим только два.

`format_spec` – ‘это строка, содержащая некоторые значения, которые указывают как будет отформатировано значение **`value`** (в нашем случае вещественное число).

Спецификация вида точка число буква f указывает сколько знаков после точки будет выведено.

Например

```
a = 234.123456789
print(format(a,'.5f'))
print(format(a,'.0f'))
print(format(a,'.1f'))
print(format(a,'.4f'))
print(format(a,'.13f'))
print(format(a,'.14f'))
print(format(a,'.15f'))
print(format(a,'.20f'))
```

получим

```
234.12346
234
234.1
234.1235
234.1234567890000
234.12345678899999
234.123456788999988
234.12345678899998802081
```

Заметим, что если знаков после точки меньше, чем в исходном числе, то последняя цифра округляется, если больше, то дописываются нули. Однако при увеличении количества знаков после точки, возможно некоторое отклонение, как в трех последних строках.

Спецификация вида точка число буква g указывает сколько всего значащих цифр будет выведено.

Например

```
a = 234.123456789
print(format(a,'.5g'))
print(format(a,'.0g'))
print(format(a,'.1g'))
print(format(a,'.2g'))
print(format(a,'.3g'))
print(format(a,'.4g'))
print(format(a,'.13g'))
print(format(a,'.14g'))
print(format(a,'.15g'))
print(format(a,'.20g'))
```

получим

```
234.12
2e+02
2e+02
2.3e+02
234
234.1
234.123456789
234.123456789
234.123456789
234.12345678899998802
```