Самое важное

Оператор return

Оператор return позволяет «вернуть» какой-либо объект из функции. Чтобы понять, что это значит, вернёмся к примеру, который мы использовали в прошлом модуле.

```
def foo(x):
    print(x)
    return x — но добавим в этот раз return x.

x = 10
x = foo(5) — приравняем результат выполнения функции переменной ‘x’, созданной снаружи
print(x)
```

Теперь запустим код и проследим за тем, как Python его выполняет.

1. Python создаёт функцию foo, на этом этапе создаётся переменная foo и в неё записывается наша функция. Теперь, в коде ниже, Python будет понимать, что, обращаясь к foo, мы пытаемся вызвать эту функцию.
2. Python создаёт переменную ‘x’ и записывает туда 10.
3. Python вызывает функцию foo с параметром x=5. На данном этапе этот икс никак не влияет на ‘x’, созданный снаружи.

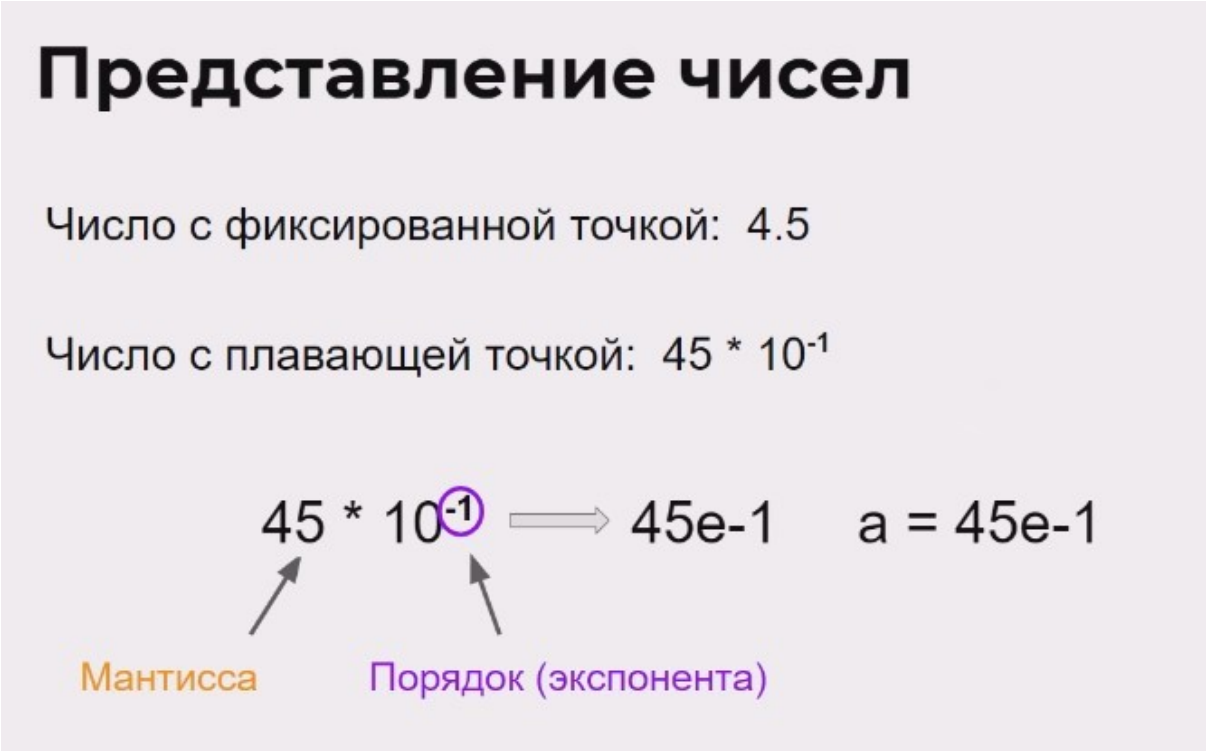
После создания своей локальной версии x (x = 5) Python выполняет вложенный код функции print(x) — распечатывает значение своего параметра ‘x’ (в консоль выведется 5) и выполняет return x — возврат значения переменной ‘x’, то есть возврат числа 5.

4. Функция выполнилась и вернула число 5, это число мы записываем в переменную ‘x’.
5. Операция печати ‘x’, созданного вне функции, print(x) распечатает не 10, а 5.

Таким образом получилось изменить переменную ‘x’, которая ранее оставалась неизменной (в примере прошлого модуля).

Экспоненциальная запись вещественных чисел
Способ записи/представления вещественных чисел, при котором указывается отдельно целая часть числа и его порядок, иногда такой способ ещё называется «научным» представлением чисел.

Как это работает?



Особенности работы с вещественными числами
Компьютер имеет ограниченное количество памяти для представления чисел. Из этого ограничения вытекает проблема, связанная с хранением некоторых дробных чисел.

Например число 0.3333…, в котором тройки можно продолжать до бесконечности, что будет с каждым разом всё более точно представлять итоговое число, не поместится полностью в памяти компьютера.

Для хранения подобных чисел компьютеру придётся использовать ограничение в 16 символов после запятой, то есть наша бесконечная дробь 0.33333333333333333333333… будет записана как 0.3333333333333333.

То есть в итоге число в компьютере не будет обладать максимальной точностью. Подобные неточности могут приводить к необычным результатам при работе с вещественными числами, поэтому в операциях с вещественными числами приходится прибегать к хитростям:

Так, например, вместо сравнения *1.1 + 2.2. == 3.3* можно записать *abs(c - d) < 1e-15,* что, по сути, значит разность чисел (по модулю) меньше самого маленького доступного числа.

Ещё один пример работы с иррациональными числами — ограничение точности вычислений:

```python
precision = float(input('Точность: '))

result = 0
i = 0
addMember = 1

while addMember > precision:
  addMember = 1 / math.factorial(i)
  result += addMember
  i += 1

print('Результат:', result)
print('Константа:', math.e)
```

В этой задаче заранее указывается число, которое будет задавать порог точности.
В процессе вычисления считаем по одному элементу последовательности до тех пор, пока новый элемент не окажется меньше заданного порога точности.
Если это произойдёт, прервём цикл и выведем результат.

Не допускай следующих ошибок!

Если вам предстоит работать с вещественными числами, постарайтесь заранее определить необходимый порог точности для вашей задачи. Помните, что чем меньше этот порог и чем точнее числа вам нужны, тем больше ресурсов потребуется для работы с этими числами.

При этом большая точность не всегда является необходимостью. Если, например, у вас есть задача по работе с валютой (допустим, рублями/копейками), то вам не обязательно использовать точность 10–15 знаков после запятой, ведь вы не сможете разделить одну копейку на более мелкие части.