Программа Python.
2021-2022 год.

Знакомство с функцией случайных чисел randint()

Задача: Вывести 5 случайных чисел от 0 до 10

```
from random import randint # Достаем функцию randint() из библиотеки random (в библиотеке random есть много функций и randint() в том числе. Но нам нужна ниже в программе только функция randint() поэтому такой синтаксис для использования в программе только функции randint().
for i in range(5):
    print(randint(0,10)) # Выводит случайным образом число от 0 до 10 (включительно!) Очень важно запомнить, что функция randint() выдает случайные числа из диапазона, который указан в скобках, при этом границы диапазона тоже попадают в случайную выдачу.
```

```
5
0
6
0
10
```

Чистый код программы без комментариев:

```
from random import randint
for i in range(5):
    print(randint(0,10))
```

Еще одна задача: Вывести среднее арифметическое значение случайных чисел от 0 до 100. Среднее арифметическое число - это число, получаемое при делении суммы чисел на их количество. Пусть среднее арифметическое значение вычисляется среди 100 случайных чисел.

```
from random import randint
sum = 0
```

```
for i in range(100): # в цикле for число 100 - это количество чисел, участвующих в вычислении среднего арифметического значения.
    sum += randint(0,100) # здесь число 100 - это правый диапазон чисел для функции randint()

print(sum / 100)
```

Видим, что цифра 100 в коде программы часто используется, поэтому можем для цифры 100 завести переменную.

```
from random import randint
sum = 0
n = 100
for i in range(n):
    sum += randint(0,n)

print(sum / n)
```

Теперь, изменяя значение переменной n (только в одном месте программы), мы можем получать среднее арифметическое значение **случайных чисел** в любом диапазоне.

А теперь немного поправим код:

```
from random import randint
sum = 0
n = 100
for i in range(n):
    sum += randint(0, 100) # Установим фиксированный диапазон случайных чисел от 0 до 100

print(sum / n)
```

Получаем какой-то случайный ответ:
48.55

А теперь что произойдет с ответом, **если n = 100000**?

```
from random import randint
sum = 0
n = 100000
for i in range(n):
    sum += randint(0, 100)
```

```
print(sum / n)
```

Получаем какой-то случайный ответ:
49.9618

Получается, чем больше у нас будет итераций (циклов), тем ближе мы будем приближаться к результату **50.** В идеале при количестве итераций (циклов) = **+ бесконечность,** ответом программы будет ровное число **50.**

Можно при помощи нашей программы подсчитать среднее арифметическое значение очков при выбрасывании игрового кубика:

```
from random import randint
sum = 0
n = 100000
for i in range(n):
    sum += randint(1, 6)

print(sum / n)
```

Ответ:
3.4978 # Значение очень близкое к 3.5

Также можно подсчитать среднее арифметическое у подбрасывания монетки (орел или решка).

```
from random import randint
sum = 0
n = 100000
for i in range(n):
    sum += randint(1, 2)

print(sum / n)
```

Ответ:
1.50276 # Значение очень близкое к 1.5

Можно подсчитать среднее арифметическое при выбрасывании двух кубиков:

```
from random import randint
```

```
sum = 0
n = 100000
for i in range(n):
    sum += randint(1, 6) + randint(1, 6)

print(sum / n)
```

Ответ:
7.00261 # Значение очень близкое к 7

Задача: Напечатать 10 чисел арифметической прогрессии начиная со значения = 1 с шагом +5.

```
number = 1
for i in range(10):
    print(number, end=" ")
    number += 5
```

Ответ: **1 6 11 16 21 26 31 36 41 46**

Задача: Напечатать 10 чисел геометрической прогрессии от 1 с шагом 3.

```
number = 1
for i in range(10):
    print(number, end=" ")
    number *= 3
```

Ответ: **1 3 9 27 81 243 729 2187 6561 19683**

Как мы можем заметить это степени числа 3:
$3^0, 3^1, 3^2, ...3^9$

Задача: Пользователь вводит любую строку (слово). Программа печатает каждую букву строки (слова) в столбец.

```
str = input()
for i in range(len(str)):
    print(str[i])
```

Результат:

Привет
П
р
и
в
е
т

Но для языка Python этот код не совсем корректный. Есть в языке Python еще одна конструкция для цикла for:

```
str = input()
for letter in str:
    print(letter)
```

Читается код следующим образом: Для каждой буквы (каждого символа) в строке напечатать эту букву (символ).

Для каждого символа (символ помещается в переменную letter) в объекте str напечатать эту переменную letter.

Получается, что Python берет переменную letter и присваивает сначала ей значение первого символа строки str. Затем переписывает значение переменной letter на значение второго символа строки, и так далее до последнего элемента (символа) строки (объекта, если в общем виде рассматривать конструкцию for)

Это новая версия цикла for. Эта версия хороша тем, что не нужно теперь использовать в написании кода программ индекс (i).

Но конечно не все задачи можно решить с помощью этой новой версии цикла for.

Поэтому будем пользоваться **и старой и новой версиями цикла for**, в зависимости от поставленных задач.

Результат аналогичный верхней программе, но текста программы гораздо меньше:

Привет
П
р

и
в
е
т

Есть еще один способ выполнить поставленную выше задачу даже **без цикла for или while,** с помощью метода работы со строками (повторение материала):

```
str = input()
print(str.replace('', '\n'))
print('END')
```

Результат:

```
Привет

П
р
и
в
е
т

END
```

Здесь нужно понять, что когда Python анализирует строку str он видит между буквами пустую строку! Добавим пару строк кода:

```
str = input()
print('Start')
print(str.replace('', '\n'))
print('END')
```

```
Привет
Start

П
р
и
в
е
т
```

END

При этом, как видим, не только между буквами Python видит пустую строку, но и даже перед первым символом строки и после последнего символа строки.

Что-то приблизительно вот так:
'','П','','р','','и','','в','','е','','т',''

Можно посмотреть вывод в одну строку, так заметнее:

```
str = 'Привет'
#print('Start')
#print(str.replace('', '\n'))
#print('END')
print(str.replace('', '*'))
```

*П*р*и*в*е*т*

Тема: сравнение строк в Python.

Строка в Python – **это набор символов**, находящийся в кавычках.

Язык программирования позволяет проводить операции сравнения строк, для этого используются следующие операторы: **<, <=, ==,!=, >, >=.**

Сравнение строк имеет **ряд своих особенностей**, отличающихся от сравнения чисел.

Рассмотрим сравнение Больше / Меньше.

Для сравнения больше или меньше используются как для строк, так и для чисел, операторы сравнения **больше - (>)** или **меньше (<).**

Пример:

```
print('apple' > 'banana')
```

False

Если мы поставим знак меньше - (<):

print('apple' < 'banana')

True

Правило сравнения строк в Python:

При сравнении символов или строк, Python конвертирует каждый символ по отдельности в их соответствующие порядковые значения, после чего последовательно сравнивает символы слева направо по их порядковому значению.

Увеличение порядкового значения начинается слева направо. Самый правый символ имеет максимальное значение порядкового номера.

Где же узнавать порядковый номер символа?
Как называется эта таблица?

Это таблицы кодировок символов **ASCII** или **Unicode.**

ASCII - это таблица символов, содержащая 128 символов.

Кодировка ASCII была разработана в Америке, поэтому стандартная кодировочная таблица обычно включает в себя только английский алфавит от A до Z с цифрами от 0 до 9 и некоторые специальные символы, что в общей сложности составляет около 128 символов.

Чтобы появилась возможность использовать языки других стран в таблице, кроме английского, пришлось создавать другую таблицу **Unicode.**

Unicode — это основной стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира.

Unicode применяется везде, где есть текст.

Информация на страницах в социальных сетях, записи в базах данных, компьютерные программы и мобильные приложения — всё это работает с использованием Unicode.

В Unicode постепенно добавлялись новые языки и символы.

По данным **wikipedia** в 14.0 версии таблицы от сентября 2021 года содержится уже 144 697 символов.

Возвращаемся к сравнению строк:

Посмотрим снова на наш пример:

```
print('apple' < 'banana')
```

```
True
```

Логично предположить, что слово **apple** находится перед **banana** (**значит меньше**), потому что буква **'a'** идет перед **'b'**, что и дает такой результат.

Теперь давайте рассмотрим еще один случай сравнения строк.

Проверим равенство apple и Apple:

```
print('apple' == 'Apple')
```

```
False
```

```
print('apple' > 'Apple')
```

```
True
```

Дело в том, что одинаковые буквы в разных регистрах считаются разными символами. Компьютер различает символы по присвоенным им уникальным значениям.

В нашем случае, латинская буква **'A'** имеет значение 65, строчная **'a'** – 97.

Узнать уникальное значение любого символа можно с помощью функции **ord()**.

Пример:

print(ord('A'), ord('a'))

65 97

Как уже выше было сказано, при сравнении символов или строк, Python конвертирует символы в их соответствующие порядковые значения, после чего сравнивает слева направо.

В примере выше **'a'** больше по порядковому значению чем **'A'**, соответственно **'apple'** больше чем **'Apple'.**

Существует функция **chr()**, преобразовывающая наоборот порядковое значение в символ.

Пример:

print(chr(1040))

А - это русская большая буква А.

Многие думают, что Python складывает сразу все индексы символов из таблицы **Unicode**, которые содержатся в строке, а потом сравнивает полученные значения. Но это не так:

Это можно проверить:

Пример:

print('ab' > 'aa')

print(ord('a') + ord('b'), 'больше', ord('a') + ord('a'),' - это', ('ab' > 'aa'))

True
195 больше 194 - это True

Кажется, что утверждение о том, что Python при сравнении строк складывает суммы индексов по каждой строке верное утверждение.

Но если мы изменим наши строки в сравнении:

print('aba' > 'aab')

print(ord('a') + ord('b') + ord('a'), 'больше', ord('a') + ord('a') + ord('b'),' - это', ('aba' > 'aab'))

True
292 больше 292 - это True - это не верно!

Набор символом в строках одинаковый, поэтому сумма индексов одинаковая: 292

Но выражение **292 > 292 это никак не может быть True**!

А строки у нас все-таки сравнились верно, и оказалось, что строка **'aba'** больше строки **'aab'.**

Почему?

Потому, что второй символ строки **'aba'** больше по индексу второго символа строки **'aab'.**

Закрепим наши знания по сравнения строк еще одним примером.

Пример:

print('abcd' < 'abce')

True

В примере мы видим сравнение строк, у которых четыре символа.

Отличие между строк только в последнем символе.

Давайте разберемся, почему строка **'abcd'** меньше строки **'abce'**

Сначала сравнивается порядковый номер первых символов у строк между собой.

'a' < 'a' так как буквы равны между собой, то Python пропускает эти буквы и переходит к сравнению следующей пары символов между собой по их порядковым номерам.

'b' < 'b' - тоже пропускаются
'c' < 'c' - тоже пропускаются

'd' < 'e' - эти символы не равны, поэтому возможно произвести их сравнение больше или меньше.

В нашем примере мы проверяем **'d' < 'e'**

Узнаем порядковые номера этих символов:

```
print(ord('d'), ord('e'))
```

```
100 101
```

Получается, если заменить символы порядковыми номерами, то выражение по сравнению будет выглядеть следующим образом:

```
100 < 101 ==> True
```

Соответсвенно:

```
'd' < 'e' ==> True
```

Поэтому в результате сравнения двух строк:

'abcd' < 'abce' мы в результате проверки последовательно каждого символа слева направо получаем логическое значение **True.**

Другими словами, мы смогли определить какая строка меньше из двух по последнему символу, потому что первые три символа равны.

Теперь еще одно правило сравнения строк между собой:

Пример: Добавим к левой строке в конец самый большой по порядковому номеру английского алфавита символ **'z'**, а к правой строке в конец добавим самый маленький символ по порядку **'a'**.

```
print('abcdz' < 'abcea')
```

```
True
```

В результате сравнения строк мы видим, что результат не изменился строка **'abcdz'** по-прежнему остается меньше строки **'abcea'**.

Из этого примера можно сделать вывод: Как только при сравнении строк пара символов дают логический ответ (True или False), то строки считаются проверенными на сравнение.

Мы получили уже ответ, какая строка больше или меньше, и нет уже необходимости сравнивать следующие пары символов в строках, так как ответ уже есть.

Мы знаем, какая строка меньше или больше.
Сравнение последующих символов не изменит уже логическое значение, которое было получено.