Строки. Ввод и форматирование.

ТЕКСТ (СТРОКИ) (STR, STRING): любой текст внутри одинарных или двойных кавычек.

Важно: целое число в кавычках - это тоже строка, не целое число.

In [21]:
```
x = 'text'
print(type(x))
print(type('Hello, world!'))
print(type('2'))
```

```
<class 'str'>
<class 'str'>
<class 'str'>
```

Если попробовать сложить число-строку и число-число, то получим ошибку.

In [22]:
```
print('2' + 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-81ac1f9e7aa0> in <module>()
----> 1 print('2' + 3)

TypeError: must be str, not int
```

Еще раз обратим внимание на текст ошибки. *TypeError: must be str, not int* - TypeError означает, что данная команда не может быть исполнена из-за типа какой-то из переменных. В данном случаем видим, что что-то должно быть "str", когда оно типа "int". Давайте попробуем сделать 3 тоже строкой. Кстати, неважно какие вы используете кавычки (только если это не текст с кавычками внутри), главное, чтобы открывающая и закрывающая кавычки были одинаковые.

In [24]:
```
print('2' + "3")
```

```
23
```

Такая операция называется контакенация (слиянием) строк.

Можно и умножить строку на число можно. Такая операция повторит нам строку заданное количество раз.

In [1]:
```
'2' * 3
```

Out[1]:
```
'222'
```

Операции со строками тоже работают, если строки лежат внутри переменных.

In [2]:
```
word1 = 'John'
word2 = ' Brown'
print(word1 + word2)
word3 = word1 + word2 # можем результат контакенации положить в новую переменную
print(word3)
```

```
John Brown
John Brown
```

В прошлом блокноте мы преобразовывали строки в числа с помощью функции int().

In [36]:
```
print(2 + int('2512')) # нет ошибки!
```

```
2514
```

С помощью функции str() мы можем превращать другие типы данных в строки.

In [3]:
```
print('abs' + str(123) + str(True))
```

```
abs123True
```

Обратите внимание: эти функции (str, int, float и другие) не меняют тип самих данных или переменных, в которых они хранятся. Если мы не перезапишем значение, то строка станет числом только в конкретной строчке команды, а ее тип не изменится.

In [38]:
```
a = '2342123'
print(type(a))
print(2 + int(a))
print(type(a))
```

```
<class 'str'>
2342125
<class 'str'>
```

In [39]:
```
a = int(a) # перезаписываем значение переменной
print(type(a)) # теперь изменился и тип
```

```
<class 'int'>
```

Ввод данных. input()

Познакомимся с функцией input() - ввод. Это функция позволяет нам просить пользователя что-нибудь ввести или принять на ввод какие-то данные.

In [5]:
```
x = input('Как вас зовут?')
print(x)
```

```
Олег
```

'Как вас зовут?' - аргумент функции input(), который будет отображаться в строке ввода. Он необязателен, но он может быть полезен для вас (особенно, если в задаче требуется ввести больше одной строки данных и нужно не запутаться).

Обратите внимание, что функция input() все сохраняет в строковом формате данных. Но мы можем сделать из числа-строки число-число с помощью функции int(). Введите "1" в обоих случаях и сравните результат.

In [6]:
```
print(type(input()))
print(type(int(input())))
```

```
<class 'str'>
<class 'int'>
```

Нужно следить за тем, какие данные вы считываете и менять их при необходимости. Сравните два примере ниже.

In [2]:
```
x = input('Input x value: ')
y = input('Input y value: ')
print(x + y) # произошла склейка строк, по умолчанию input() сохраняет значение типа
```

```
23
```

In [3]:
```
x = int(input('Input x value: '))
y = int(input('Input y value: '))
print(x + y) # вот теперь произошло сложение чисел
```

```
5
```

(∩ ` -´)⊃–☆˚.*･｡˚ Задача

Pig Latin 1

Pig Latin - вымышленный язык, который в конец каждого слова подставляет 'yay' (на самом деле оригинальные условия задачи несколько хитрее, но мы будем к ней возвращаться по ходу курса.

Формат ввода
Пользователь вводит слова.

Формат вывода Слово на Pig Latin.

Примеры

Тест 1
Ввод:
apple

Вывод:
appleyay

In [2]:
```
# (∩ `-´)⊃–☆˚.*･｡˚
word = input('Введите слово: ')
print(word+'yay')
```

```
appleyay
```

(∩ ` -´)⊃–☆˚.*･｡˚ Задача

x потвторенное x раз в квадрате

Вводится целое число X. Повторите его X раз и возведите полученное число в квадрат. Напечатайте получившееся.

Формат ввода
Целое число

Формат вывод

Ответ на задачу

Примеры

Тест 1

Ввод:

2

Вывод:

484

In [6]:
```
x = input()
print((int(x*int(x)))**2)
```

```
484
```

(∩ ` -´)⊃–☆゚.*･｡ﾟ Задача

Хитрости умножения

Для умножения двузначного числа на 11 есть интересная хитрость: результат произведения можно получить если между первой и второй цифрой исходного числа вписать сумму этих цифр. Например, 15 *11 = 1 1+5 5 = 165 или 34* 11 = 3 3+4 4 = 374. Реализуйте программу, которая умножала бы двузначные числа на 11, используя эту хитрость. Пользоваться оператором умножения нельзя.

Формат ввода:

Вводится двузначное число.

Формат вывода:

Программа должна вывести ответ на задачу.

Тест 1

Пример ввода:

15

Вывод:

165

Тест 2

Пример ввода:

66

Вывод:

726

In [1]:
```
# ваше решение здесь
x = input()

n1 = int(x[0])
n2 = int(x[1])

if n1 + n2 < 10:
```

```
    print(str(n1) + str(n1+n2) + str(n2))
else:
    print(str(n1 + 1) + str((n1 + n2) % 10) + str(n2))
```

```
78
858
```

Форматирование строк (string formatting)

А теперь посмотрим на то, как подставлять значения в уже имеющийся текстовый шаблон, то есть форматировать строки. Чтобы понять, о чём идет речь, можно представить, что у нас есть электронная анкета, которую заполняет пользователь, и мы должны написать программу, которая выводит на экран введенные данные, чтобы пользователь мог их проверить.

Пусть для начала пользователь вводит свое имя и возраст.

In [3]:
```
name = input("Введите Ваше имя: ")
age = int(input("Введите Ваш возраст: ")) # возраст будет целочисленным
```

Теперь выведем на экран сообщение вида

```
Ваше имя: имя . Ваш возраст: возраст .
```

Но прежде, чем это сделать, поймем, какого типа будут значения, которые мы будем подставлять в шаблон. Имя (переменная name) – это строка (string), а возраст (переменная age) – это целое число (integer).

In [4]:
```
result = "Ваше имя: %s. Ваш возраст: %i." % (name, age)
print(result)
```

```
Ваше имя: Олег. Ваш возраст: 32.
```

Что за таинственные %s и %i? Все просто: оператор % в строке указывает место, на которое будет подставляться значение, а буква сразу после процента – сокращённое название типа данных (s – от string и i – от integer). Осталось только сообщить Python, что именно нужно подставлять – после кавычек поставить % и в скобках перечислить названия переменных, значения которых мы будем подставлять.

Для тех, кто работал в R: форматирование строк с помощью оператора % – аналог форматирования с помощью функции sprintf() в R.

Примечание: не теряйте часть с переменными после самой строки. Иначе получится нечто странное:

In [5]:
```
print("Ваше имя: %s. Ваш возраст: %i.")
```

```
Ваше имя: %s. Ваш возраст: %i.
```

Важно помнить, что если мы забудем указать какую-то из переменных, мы получим ошибку (точнее, исключение): Python не будет знать, откуда брать нужные значения.

In [6]:
```
print("Ваше имя: %s. Ваш возраст: %i." % (name))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-2343aa81e176> in <module>()
----> 1 print("Ваше имя: %s. Ваш возраст: %i." % (name))

TypeError: not enough arguments for format string
```

Кроме того, создавая такие текстовые шаблоны, нужно обращать внимание на типы переменных, значения которых мы подставляем.

In [7]:
```
print("Ваше имя: %s. Ваш возраст: %s." % (name, age)) # так сработает
```

```
Ваше имя: Олег. Ваш возраст: 32.
```

In [8]:
```
print("Ваше имя: %i. Ваш возраст: %s." % (name, age)) # а так нет
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-9c083ea32449> in <module>()
----> 1 print("Ваше имя: %i. Ваш возраст: %s." % (name, age)) # а так нет

TypeError: %i format: a number is required, not str
```

В первом случае код сработал: Python не очень строго относится к типам данных, и поэтому он легко может превратить целочисленный возраст в строку (два %s вместо %s и %i не является помехой). Во втором случае все иначе. Превратить строку, которая состоит из букв (name) в целое число никак не получится, поэтому Python справедливо ругается.

А что будет, если мы будем подставлять не целое число, а дробное, с плавающей точкой? Попробуем!

In [9]:
```
height = float(input("Введите Ваш рост (в метрах): "))
height
```

Out[9]:
```
1.78
```

In [10]:
```
print("Ваш рост: %f м." % height) # f - от float
```

```
Ваш рост: 1.780000 м.
```

По умолчанию при подстановке значений типа float Python выводит число с шестью знаками после запятой. Но это можно исправить. Перед f нужно поставить точку и указать число знаков после запятой, которое мы хотим:

In [11]:
```
print("Ваш рост: %.2f м." % height) # например, два
```

```
Ваш рост: 1.78 м.
```

In [12]:
```
print("Ваш рост: %.1f м. " % height) # или один
```

```
Ваш рост: 1.8 м.
```

В случае, если указанное число знаков после запятой меньше, чем есть на самом деле (как в ячейке выше), происходит обычное арифметическое округление.

Рассмотренный выше способ форматирования строк – не единственный. Он довольно стандартный, но при этом немного устаревший. В Python 3 есть другой способ – форматирование с помощью метода .format(). Кроме того, в Python 3.6 и более поздних версиях появился еще более продвинутый способ форматирования строк – f-strings (formatted string literals).

F-strings очень удобны и просты в использовании: вместо % и сокращённого названия типа в фигурных скобках внутри текстового шаблона нужно указать название переменной, из которой должно подставляться значение, а перед всей строкой добавить f, чтобы Python знал, что нам нужна именно f-string.

In [16]:
```
print(f"Ваше имя: {name}. Ваш возраст: {age}. Рост: {height:.2f}")
```

```
Ваше имя: Олег. Ваш возраст: 32. Рост: 1.78
```

Альтерантивой такого кода будет следующий синтаксис. Здесь переменные вставятся в порядке "упоминания" в строке.

In [14]:
```
print("Ваше имя: {}. Ваш возраст: {}".format(name, age))
```

```
Ваше имя: Олег. Ваш возраст: 32
```

Форматирование дробей добавляем так.

In [15]:
```
print("Ваше имя: {}. Ваш возраст: {}. Рост: {:.2f}".format(name, age, height))
```

```
Ваше имя: Олег. Ваш возраст: 32. Рост: 1.78
```

Если указали формат переменной (.2f, например, ожидает float), то следим за порядком.

In [17]:
```
print("Ваше имя: {}. Ваш возраст: {}. Рост: {:.2f}".format(age, height, name))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-f42c355e5798> in <module>()
----> 1 print("Ваше имя: {}. Ваш возраст: {}. Рост: {:.2f}".format(age, height, name
))

ValueError: Unknown format code 'f' for object of type 'str'
```

Порядок заполнения можно указывать с помощью нумерации. Одну и тоже переменную можно использовать несколько раз.

In [32]:
```
print("Ваше имя: {2}. Ваш возраст: {0}. Рост: {1:.2f}. Все верно, {2}?".format(age,
```

```
Ваше имя: Олег. Ваш возраст: 32. Рост: 1.78. Все верно, Олег?
```

Также при форматировании строк можно использовать результаты вычислений и результаты работы методах (о них немного позже). name.upper() в примере делает все символы строки name заглавными.

In [36]:
```
print(f"Ваше имя: {name.upper()}. Ваш возраст: {2020-1988}. Рост в футах: {height *
```

```
Ваше имя: ОЛЕГ. Ваш возраст: 32. Рост в футах: 5.84
```

(∩ ` -´)⊃–☆゜.*･｡ﾟ Задача

Pi

Из модуля math импортируйте переменную pi. С помощью %f (первое предложение) и format (второе предложение) форматирования выведите строки из примера. Везде, где встречается число pi - это должна быть переменная pi, округленная до определенного знака после точки.

Формат вывода:

```
Значение 22/7 (3.14) является приближением числа pi (3.1416)
Значение 22/7 3.142 является приближением числа pi 3.141592653589793
```

In [25]:

```
import math
print(math.pi)
print('Значение 22/7 (%.2f) является приближением числа pi (%.4f)' % (math.pi, math.
print(f'Значение 22/7 {math.pi:.3f} является приближением числа pi {math.pi}')
```

```
3.141592653589793
Значение 22/7 (3.14) является приближением числа pi (3.1416)
Значение 22/7 3.142 является приближением числа pi 3.141592653589793
```

Функция print(): аргументы и параметры

И еще немного поговорим про форматирование вывода. Это вам пригодится, если будете решать много задача на разных сайтах вроде hackerrank, где ответ нужно выводить каким-то хитрым образом. На самом деле функция print() немного сложнее, чем то, что мы уже видели. Например, мы можем печатать одной функцией больше чем один аргумент и использовать разделители вместо пробелов.

In [26]:

```
print('2 + 3 =', 2 + 3)
z = 5
print('2 + 3 =', z)
```

```
2 + 3 = 5
2 + 3 = 5
```

У функции print есть параметры. Параметры - это такие свойства функций, которые задают значение невидимых нам аргументов внутри функции. Так существует параметр sep (separator), благодаря которому мы можем менять тип разделителя между аргументами print. В качестве разделителя может выступать любая строка.
Сравните:

In [29]:

```
print('1', '2', '3')
print('1', '2', '3', sep='.')
print('1', '2', '3', sep='')
print('1', '2', '3', sep='\n')
```

```
1 2 3
1.2.3
123
1
```

```
2
3
```

Параметр end задает то, что будет выведено в конце исполнения функции print. По умолчанию там стоит невидимый символ '\n', который осуществляет переход на новую строку. Мы можем заменить его на любую строку. Если мы хотим сохранить переход на новую строку - то обязательно прописываем наш невидимый символ внутри выражения.

In [28]:
```
print('1', '2', '3', sep='.', end='!')
print('2')  # строки слились

print('1', '2', '3', sep='.', end='!\n')
print('2')  # вывод на новой строке
```

```
1.2.3!2
1.2.3!
2
```