

ТИПЫ ДАННЫХ

Created by:

Chingiz Bapanov
Bektur Toktogulov
Marselle Nazarmatov

for ITC Bootcamp



print()

Для начала
ознакомимся с
функцией print() для
того чтобы вы могли
протестировать
свой текстовый
редактор

print() - это **функция** который выводит на экран результат кода. Это происходит в терминале (еще называют - интерпретатор).

```
print('внутри скобок пишутся данные')  
print("Hello ITC bootcamp")
```

Всё что находится внутри python файла являются данными.
Каждый из них имеет свой тип

```
Strings.py X a = 5.py  
C: > Users > Admin > Desktop > Migr-master > Strings.py  
1 print("Hello ITC bootcamp")  
2  
  
ПРОБЛЕМЫ Выходные данные Консоль отладки ТЕРМИНАЛ  
  
Windows PowerShell  
(C) Корпорация Майкрософт (Microsoft Corporation)  
  
Попробуйте новую кроссплатформенную оболочку PowerShell  
  
PS C:\Users\Admin> & C:/Users/Admin/AppData/Local/Strings.py  
Hello ITC bootcamp  
PS C:\Users\Admin>  
  
Содержимое сеанса восстановлено из 30.05.2022 в 10:00:00
```

```
itcbootcamp > proekty > flp.py > ...  
10 for i in range(15):  
11     print(i, end='|')  
12  
ПРОБЛЕМЫ Выходные данные Консоль отладки ТЕРМИНАЛ  
  
0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|  
PS C:\Users\Selçuk Dirlikli\OneDrive\Desktop\Desktop>
```

```
itcbootcamp > proekty > flp.py > ...  
10 for i in range(15):  
11     print(i, end='?')  
12  
ПРОБЛЕМЫ Выходные данные Консоль отладки ТЕРМИНАЛ  
  
0?1?2?3?4?5?6?7?8?9?10?11?12?13?14?  
PS C:\Users\Selçuk Dirlikli\OneDrive\Desktop\Desktop>
```

Типы данных- это класс

1) **string** (str) то есть строка.

все что находится в кавычках ('одинарных', "двойных", ""тройных"") являются строкой. Неизменяемый тип

2) **integer** (int) - называют целые числа.

Например 1, 62, 43, 87, 5 ... Неизменяемый тип

3) **float** (float) - это дробные числа

такие как 1.4, 55.6, 87.9 ...

Неизменяемый тип

Переменные это “контейнеры” которые содержат в себе данные. Им даются произвольные имена, но они не могут начинаться с цифр или спец. символов

Правильное создание переменной

```
ltc_bootcamp = 'python'
ltc_bootcamp = 452
ltc_bootcamp = 34.5
```

Если в названии применяются больше двух слов то, что нужно разделять их _ нижним подчеркиванием

В переменную можно вкладывать все типы данных

Неправильное создание переменной

```
76ltc_bootcamp = 'python'
ltc_bootcamp = 452
*ltc_bootcamp = 34.5
```

Начинать переменную с цифр и спец. знаков, также ставить пустые пробелы между словами.

Input("")

```
a = 18
```

```
a = input('Введите: ')
```

Существует несколько способов передачи данных в переменную
в первом случае мы просто приравниваем "а" к 18,
во втором получаем данные с помощью функции input()

input() - это функция которая помогает нам **ввести данные** в терминале

input("внутри скобок input пишется вспомогательное слово")

по умолчанию input() **всегда** получает данные в **виде string**,

а если нужно работать цифрами, числами то необходимо

переконвертировать в integer. Следующим образом:

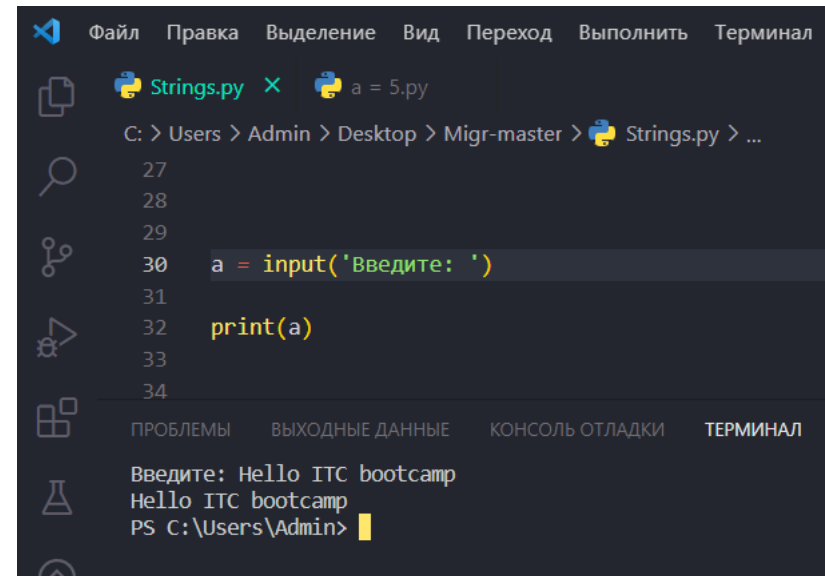
```
int(input("Введите число: ")) # таким образом можно получить integer
```

```
float(input("Введите float: ")) # таким образом можно получить float
```

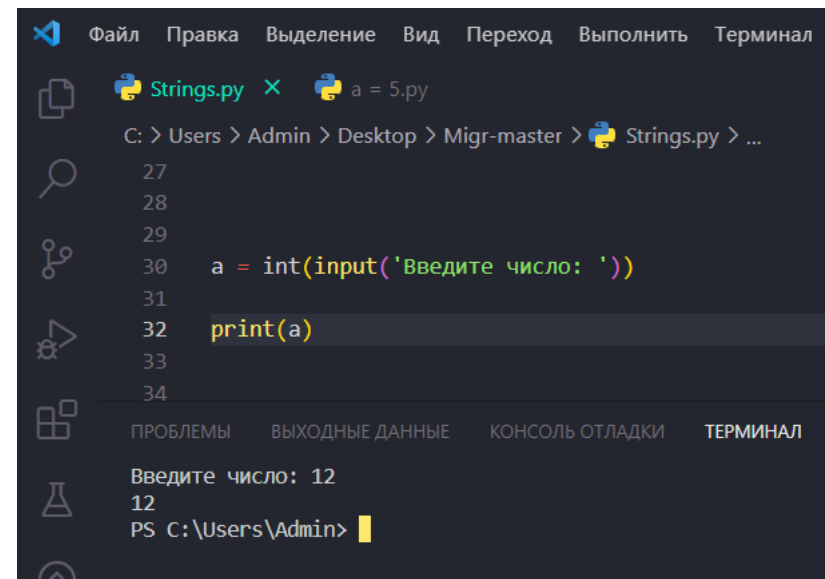
конвертация это переход из одного типа данных на другое.

```
int(input("Введите integer: ")) # таким образом можно получить integer
```

```
float(input("Введите float: ")) # таким образом можно получить float
```



```
File  Правка  Выделение  Вид  Переход  Выполнить  Терминал
Strings.py  a = 5.py
C: > Users > Admin > Desktop > Migr-master > Strings.py > ...
27
28
29
30 a = input('Введите: ')
31
32 print(a)
33
34
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ
Введите: Hello ITC bootcamp
Hello ITC bootcamp
PS C:\Users\Admin>
```



```
File  Правка  Выделение  Вид  Переход  Выполнить  Терминал
Strings.py  a = 5.py
C: > Users > Admin > Desktop > Migr-master > Strings.py > ...
27
28
29
30 a = int(input('Введите число: '))
31
32 print(a)
33
34
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ
Введите число: 12
12
PS C:\Users\Admin>
```

Операторы

Арифметические действие в Python проводятся почти таким же образом как и в арифметике.

На примере это выглядит так: « $10 + 5 = 15$ ».

В этом выражении 10 и 5 — **операнды**. Знак «+» — **оператор**.

Создадим для примера две переменные **a** и **b**

Поместим в них данные **10** и **5**:

`a = 10`

`b = 5`

+ Добавление — добавляет левый операнд к правому `a + b = 15`

- Вычитание — вычитает правый операнд из левого `a - b = 5`

***** Умножение — умножает левый операнд на правый `a * b = 50`

/ Деление — делит левый операнд на правый `a / b = 2`

% Деление по модулю — делит левый операнд на правый и возвращает остаток. С помощью него, например, можно проверять числа на четность и нечетность.

Если левый операнд меньше правого, то результатом будет левый операнд

`a % b = 0.`

****** Возведение в степень

`a ** b = 100000`

// Целочисленное деление — деление в котором возвращается только целая часть результата.

Часть после запятой отбрасывается.

`9 // 2 = 4, 9.0 // 2.0 = 4.0`

Логический тип (Булево число или Bool)

Кроме арифметических операций, со школы нам известны **операции сравнения**. Например $5 > 4$.

В языке python **часто** звучит как вопрос: «**5 больше 4?**». В данном случае ответ будет «**да**». А если спросить $5 < 4$? (5 меньше 4), то ответ очевидно будет «**нет**». **Ответ на условия да или нет являются логическим типом**. В python вместо слов да/нет применяются **True/False (Истина/ Ложь)**

Операции сравнения не имеют привязки к числам. Сравнить можно практически всё что угодно, например, строки `password == text`. Каждый раз, когда мы входим на какой-то сайт, внутри происходит сравнение введенных логина и пароля с теми, какие есть в базе. И только если они есть (True), нас пускают во внутрь (аутентифицируют).

Список операций сравнения:

- < — меньше
- <= — меньше или равно
- > — больше
- >= — больше или равно
- == — равно
- != — не равно

`password == text` — это сравнение идентичности строк, записанных в разных переменных.

Логическая операция типа $5 > 4$ или `password == text` — это выражение, и его результат — специальное значение True («истина») или False («ложь»). Это тоже является **типом данных — bool**.

`password = '12345'`

`password == text`

`text = 12345`

это переменные



ВНИМАНИЕ!

True = 1

False = 0

в операции сравнения условия пишутся слева направо, но операция протекает **справа на лево**

```
a = 12
b = 23
c = 44
```

AND

Есть три переменные которые содержать разные числа. Решили сравнить и выяснить самое большое число из них. “a” будет самым большим числом если окажется что “a” больше “b” **и** “a” больше “c”. В коде это будет выглядеть так. “a > b **and** a > c”

В данном примере a будет меньше чем b и меньше чем c. И **ПОЭТОМУ** “a > b” покажет нам “False” и “a > c” покажет нам “False”. Результат данной операции будет “False and False”. При выводе на экран (print()) мы получим результат одним словом False. А оператор and будет показывать нам “True” **только** в том случае если **оба сравнения покажут “True”**. Даже если один из сравнении будет True, то при выводе на экран результатом будет False.

```
14 a = 12
15 b = 23
16 c = 44
17
18 prosto_peremennaya = a > b and a > c
19 print(prosto_peremennaya)
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

False

```
a = 12
b = 8
c = 16
```

OR

А теперь мы хотим узнать больше ли “a” хотя бы **одной** из переменных? Для этого мы проводим сравнение используя оператор “or”(или). В коде это будет выглядеть таким образом: a > b **or** a > c.

В данном примере a > b покажет нам True(потому что 12 действительно больше 8) и a > c покажет нам False (Потому что утверждение 12 больше 16 - **неверно**).

В итоге мы имеем два ответа сравнения это True и False. Согласно правилу кода из них выйдет только один результат, В случае если добавим в условия or, тогда если **хотя бы одно** из сравнений окажется True, то результатом будет **True**.

“or” покажет **False** только в том случае если **оба** сравнения окажутся неверными.

```
14 a = 12
15 b = 8
16 c = 16
17
18 prosto_peremennaya = a > b or a > c
19 print(prosto_peremennaya)
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

True

a	b	a and b	a or b	not a
0 (False)	0 (False)	0 (False)	0 (False)	not False = True not True = False
0 (False)	1 (True)	0 (False)	1 (True)	
1 (True)	0 (False)	0 (False)	1 (True)	
1 (True)	1 (True)	1 (True)	1 (True)	

a = 30, b = 44

NOT

В процессе написания кода бывают случаи когда необходимо вызвать False для использования. (Например в случае если пользователь ввел неправильный пароль, и мы хотим предпринять действия на этот случай.)

Если мы хотим быть уверены в том, что мы **действительно** получили False применяется оператор not. Потому что оператор not показывает True **если** False, и False если True. В коде это выглядит примерно так not a < b. Так как a меньше b. Получиться результат True. Используя not мы превращаем полученный True в False. Например not a > b. Мы уже знаем то, что “a” меньше “b” и результатом выйдет False. Но так как мы поставили **перед “a” оператор not** мы получим в итоге True.

```
14 a = 30
15 b = 44
16 prosto_peremennaya = not a < b
17 print(prosto_peremennaya)
```

ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ

False

Языки программирования адаптировали все математические операции сравнения практически в неизменном виде. Единственное серьезное отличие – операторы равенства и неравенства. В математике для этого используется обычное равно `=`, но в программировании такое встречается не часто. Во многих языках символ `=` используется для присваивания значений переменным, поэтому для сравнения в Python взяли `==`.

`=` используется для присваивания переменных

`==` используется для сравнения

Кто какой будет результат True или False
`print(True or False and True and False)?`

#Строки и методы

.find и .r поиск порядок номера

```
str1 = 'Привет мир! Как дела народ?'
```

```
print(str1.find('!'))    # метод .find ищет индекс буквы. Нужно указать саму букву и на экран выведется  
номер расположение по счету. 10
```

.index поиск порядок номера часть 2

```
str1 = 'Привет мир! Как дела народ?'
```

```
print(str1.index('!'))    # данный метод выполняет ту же функцию что и find. Разница лишь в случае ошибки  
в index покажет ValueError
```

```
print(str1.rindex('н'))    # r не работает
```

поиск индекса между определенными словами

```
index = (str1.find('Как'))    # создается новая переменная = пишется список . метод в (') слово с  
которой нужно начать поиск
```

```
print(index)
```

```
index= (str1.find('народ'))    # написать ту же созданную выше переменную = пишется список . метод в  
(') слово с которой нужно закончить поиск
```

```
print(index)
```

```
index = str1.find('a', str1.find('Как'), str1.find('народ')) # написать ту же созданную выше переменную =  
пишется список . метод в (') слово с которой нужно найти, дальше как на примере
```

```
print(index)    # в принте в () только ту созданную переменную #  
результатом покажет позицию на которой находится буква(символ). В примере это 12
```

Если требуемого слова вообще нет в списке то выдаст -1

```

#.replace - замена слова в tuple
str1 = 'Привет мир! Как дела народ?'
print(str1.replace('мир', 'друг')) # Чтобы заменить слово не изменяя переменную нужно
                                     # написать комбинацию и в () через запятую написать сперва слово которую
                                     # нужно заменить вторым само слово на которую нужно заменить. На экран выйдет Привет друг!

Как дела народ?
str1=str1.replace('мир', 'друг') # но если заменить слово создав новую переменную и прописать метод в ней,
                                   # то тогда команда print (и др. применения) будет применяться к строковой переменной.

print(str1.replace(' ', ''))      # Чтобы на экран вывелся текст без пробелов нужно в указанном примере комбинации в ()
оставить пустые пробелы. Выйдет Приветдруг!Какделанарод?

# .split команда разделяет строковую переменную на несколько элементов списка
str1_array = str1.split() # Для этого нужно создать новую переменную и написать переменную текста.split() скобки оставить
пустыми
print(str1_array)         # выйдет ['Привет', 'друг!', 'Как', 'дела', 'народ?']

str2 = '222.jpg#333jpg#4444jpg#2211#jpg' # Случай когда нужно применить split это строка (файл) которая содержит фотографии,
решетка # является разделителем между jpg файлами.
str2_array = str2.split('#')           # и для того чтобы вывести на экран фото галерея нужно сделать split. Результат
['222.jpg', '333jpg', '4444jpg', '2211', 'jpg']
print(str2_array)                     # Таким образом получают разделенные элементы в одном целом новом списке дальше можно
использовать для вывода на экран или цикл ит.д.

# .join добавление разделителей файлов в список
imgs_array = ['222.jpg#333jpg#4444jpg#2211#jpg'] # Чтобы внести изменение в список нужно создать новую переменную =, написать
каким будет знак разделителя .join(название списка). Если Список в [] то выведет 222.jpg#333jpg#4444jpg#2211#jpg
imgs_str= '!0!'.join(imgs_array)          # Если без [] результат
2!0!2!0!2!0!.!0!j!0!p!0!g!0!#!0!3!0!3!0!3!0!j!0!p!0!g!0!#!0!4!0!4!0!4!0!4!0!j!0!p!0!g!0!#!0!2!0!2!0!1!0!1!0!#!0!j!0!p!0!g
print(imgs_str)

```

```

# .Lower уменьшения шрифта на экране
str1 = 'Привет мир! Как дела народ?'
print(str1.lower()) # сперва переменная str1 затем .lower() пустые скобки. Результат    привет мир! как дела народ?

# .upper увеличение шрифта на экране
str1 = 'Привет мир! Как дела народ?'
print(str1.upper()) # сперва переменная str1 затем .lower() пустые скобки. Результат    привет мир! как дела народ? ПРИВЕТ МИР!
КАК ДЕЛА НАРОД?

# .count подсчет количества букв или символов в списке
str1 = 'Привет мир! Как дела народ?'
print(str1.count('и'))      # сперва переменная str1 затем .count() в скобках в '' указать букву/цифру/символ покажет
количество встречающегося в списке. Результат 2
print(str1.count('д?'))     # или можно задать в ('') несколько знаков которые расположены друг за другом.
                           # Эти несколько знаков будут считаться как целое. при обнаружении покажет сколько раз встречался
в списке. Результат 1

# Len Функция подсчета количества символов
str1 = 'Привет мир! Как дела народ?'
print(len(str1)) # пишется перед переменной как в примере. Результат 27 букв

# .isalpha для того чтобы проверить строковую переменную на наличие одних БУКВ без знаков и символов и пробелов
str2 = 'RRRWWWTTTT'
print(str2.isalpha()) # результат True . Если есть цифры, пробелы и др. кроме букв то покажет False

# .isdigit для того чтобы проверить строковую переменную на наличие одних ЦИФР без знаков и букв и пробелов
str2 = '09878887766'
print(str2.isdigit()) # результат True . Если есть буквы, пробелы и др. кроме цифр то покажет False
                     # Так можно проверять списки на наличие ошибок на наличие букв или цифр

```

```
# .startswith Для проверки начинается ли строка с указанной подстроки (слова)
yesno = str1.startswith('Привет') # Нужно создать новую переменную = название списка. startswith в (') написать слово с
которой начинается список.
print(yesno) # результат выйдет True или False

# та же проверка другим способом через .find
yesno = str1.startswith('Привет')
print(str1.find('Привет')== 0) # т.е. мы в принте через find находим индекс слова затем утверждаем то что найденное слово
находится на 0 позиции (== 0) т.е.
# результат True

#endwith ля проверки заканчивается ли строка с указанной подстроки (слова)
yesno = str1.endswith('народ?') # Нужно создать новую переменную = название списка. startswith в (') написать слово с которой
начинается список.
print(yesno) # результат True

# та же проверка другим способом через .find
yesno = str1.endswith('Привет')
print(str1.find('народ?')== len(str1)-len('народ?')) # т.е. мы в принте через find находим индекс слова затем утверждаем то
что найденное слово является (равен) разницей
# указанное слово (народ) индекс которое ищем отнять от количество слов списка.
# результат True
```

```

# Позиционный .format() Чтобы вставлять в нужные тексты имена и даты которые заложены в отдельных переменных нужен данный метод.
name = 'Семен'           # Три разные переменные содержат разные типы данных
middle_name = 'Семенович' # name, middle_name это string
age = 40                 # age является float

profile = """Дорогой {1} {0}, поздравляем Вас с Вашим {2} летием!""".format(name, middle_name, age)
# Открывается тремя """ затем создается новая переменная = пишется текст(код) где должны использоваться
переменные затем ставятся фигурные скобки {2}
print(profile)           # внутри номер индекса (по порядку и кол-ву переменных) в тех местах в тексте(коде) где должны быть
нужные переменные, затем закрываются тремя """ .format(в скобках через , переменные)
# на экран выйдет Дорогой Семен Семенович, поздравляем Вас с Вашим 40 летием!
# можно переставить в тексте скобки (получается переменные) местами {1}{0}{2} Результат Дорогой
Семенович Семен, поздравляем Вас с Вашим 40 летием!

# Именной .format() В фигурные скобки новой переменной ставятся данные из переменных, при этом после формата нужно
написать что переменная=переменной и так с каждой.
name = 'Семен'           # Три разные переменные содержат разные типы данных
middle_name = 'Семенович' # name, middle_name это string
age = 40
profile = """Дорогой {name} {middle_name}, поздравляем Вас с Вашим {age} летием!""".format(name=name, middle_name=middle_name,
age=age) # Можно менять местами ничего не измениться
print(profile)           # Результат тот же Дорогой Семен Семенович, поздравляем Вас с Вашим 40 летием!
# Можно в скобах {} сократить название переменных {n}{m}{a} код все равно будет работать.

```

Example	Data Type
<code>x = "Hello World"</code>	<code>str</code>
<code>x = 20</code>	<code>int</code>
<code>x = 20.5</code>	<code>float</code>
<code>x = 1j</code>	<code>complex</code>
<code>x = ["apple", "banana", "cherry"]</code>	<code>list</code>
<code>x = ("apple", "banana", "cherry")</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = {"name": "John", "age": 36}</code>	<code>dict</code>
<code>x = {"apple", "banana", "cherry"}</code>	<code>set</code>
<code>x = frozenset({"apple", "banana", "cherry"})</code>	<code>frozenset</code>
<code>x = True</code>	<code>bool</code>
<code>x = b"Hello"</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>
<code>x = None</code>	<code>NoneType</code>