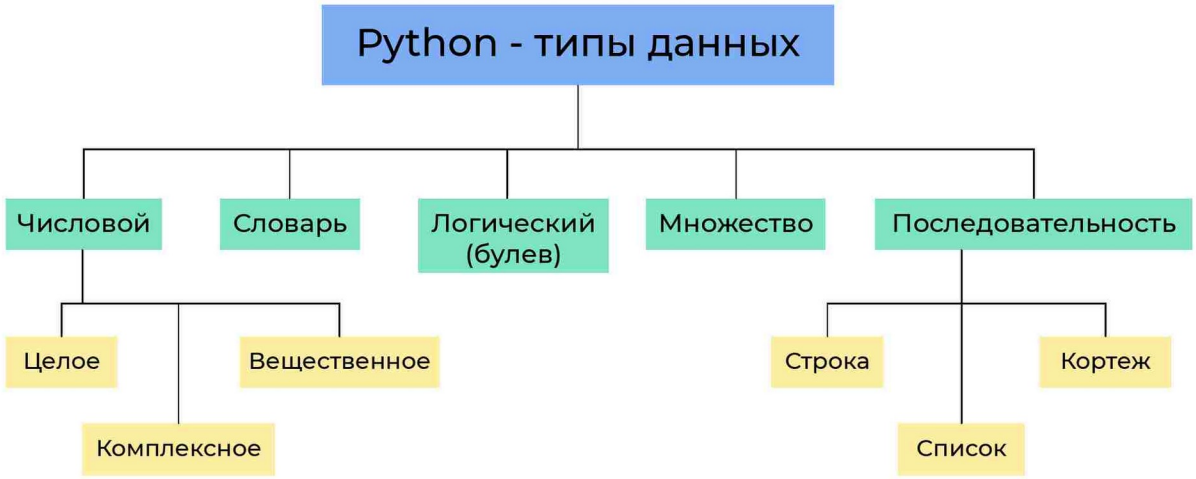
bool

Тип данных bool

Типы данных в Python

В предыдущем курсе мы изучали примитивные типы данных, такие как int, float, str, list и т.д. В их числе мы упоминали и о логическом типе данных, однако вскользь. Давайте закроем пробелы текущим уроком, который посвящен целиком и полностью логическому типу данных, который в Python представлен типом bool .



Логический тип данных

Логический тип данных (булев тип, Boolean) — примитивный тип данных в информатике, принимающий два возможных значения, иногда называемых истиной (`True`) и ложью (`False`). Присутствует в подавляющем большинстве языков программирования как самостоятельная сущность или реализуется через численный тип данных. В некоторых языках программирования за значение "истина" принимается 1, за значение "ложь" — 0.

Название типа `Boolean` получило в честь английского математика и логика Джорджа Буля, среди прочего занимавшегося вопросами математической логики в середине XIX века.

Логический тип данных в Python

Логические значения `True` (истина) и `False` (ложь) представляют тип данных `bool`. У этого типа только два возможных значения и два соответствующих литерала: `True` и `False`.

Мы активно использовали логический тип данных, когда работали с флагами:

flag = False

или когда использовали условный оператор `if-else`:

a = 100

```
b = 17

if b > a:
  print('b больше a')
else:
  print('b не больше a')
```

Результатом логического выражения `b > a` является булево значение, в данном примере `False`, так как значение в переменной `b` меньше значения в переменной `a`.

Логические выражения можно использовать не только в условном операторе.

Следующий программный код:

```
print(17 > 7)
print(17 == 7)
print(17 < 7)
```

выведет:

```
True
False
False
```

Логический тип данных – основа информатики.

Логические операторы в Python

Для создания произвольно сложных логических выражений (условий) мы используем три логические операции:

- **и (`and`);**
- **или (`or`);**
- **не (`not`).**

Логические операции используют операнды со значениями `True` и `False` и возвращают результат также с логическими значениями. Определённые для объектов типа `bool` операторы (`and`, `or`, `not`) известны как логические операторы и имеют общеизвестные определения:

- `a and b` даёт `True`, если оба операнда `True`, и `False`, если **хотя бы один** из них `False`;
- `a or b` даёт `False`, если оба операнда `False`, и `True`, если **хотя бы один** из них `True`;
- `not a` даёт `True`, если `a` имеет значение `False`, и `False`, если `a` имеет значение `True`.

Следующий программный код:

```
a = True
b = False
```

```
print('a and b is', a and b)
print('a or b is', a or b)
print('not a is', not a)
```

выведет:

```
a and b is False
a or b is True
not a is False
```

Запомните: приоритет оператора not выше, чем у оператора and, приоритет которого, в свою очередь, выше, чем у оператора or.

Булевы значения как числа

Логические значения в Python можно трактовать как числа. Значению `True` соответствует число 1, в то время как значению `False` соответствует 0. Таким образом, мы можем сравнить логические значения с числами:

Следующий программный код:

```
print(True == 1)
print(False == 0)
```

выведет:

```
True
True
```

Мы можем также применять арифметические операции к логическим значениям.

Следующий программный код:

```
print(True + True + True - False)
print(True + (False / True))
```

выведет:

```
3
1.0
```

Возможность трактовать булевы выражения как числа на практике используется не так часто. Однако есть один прием, который может оказаться полезным. Поскольку `True` равно 1, а `False` равно 0, сложение логических значений вместе – это быстрый способ подсчета количества значений `True`. Это может пригодиться, когда требуется подсчитать количество элементов, удовлетворяющих условию.

Следующий программный код:

```
numbers = [1, 2, 3, 4, 5, 8, 10, 12, 15, 17]
res = 0

for num in numbers:
    res += (num % 2 == 0)
```

print(res)

выведет количество **четных элементов** списка `numbers`, то есть число 5.

Примечания

Примечание 1. Вместо избыточного кода:

```
if flag == True:
```

программисты обычно пишут код:

```
if flag:
```

Аналогично, вместо кода

```
if flag == False:
```

программисты обычно пишут код:

```
if not flag:
```

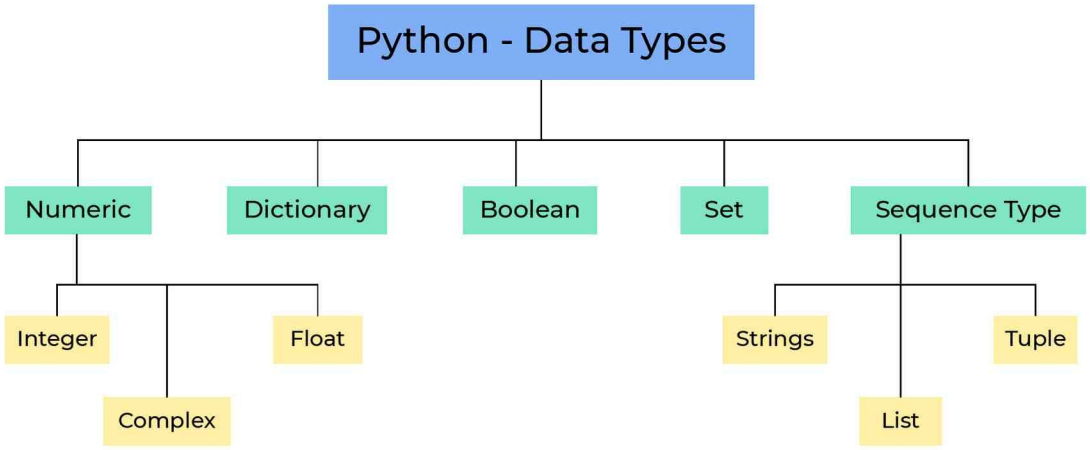
Примечание 2. Операторы `and` и `or` **ленивые**:

- при вычислении логического выражения `x and y`, если `x == False`, то результат всего выражения `x and y` будет `False`, так что `y` **не вычисляется**;
- при вычислении логического выражения `x or y`, если `x == True`, то результат всего выражения `x or y` будет `True`, и `y` **не вычисляется**.

Примечание 3. Математическая теория булевой логики определяет, что никакие другие операторы, кроме `not`, `and` и `or`, не нужны. Все остальные операторы на двух входах могут быть указаны в терминах этих трех операторов. Все операторы на трех или более входах могут быть указаны в терминах операторов двух входов.

Фактически, даже наличие пары `or` и `and` избыточно. Оператор `and` может быть определен в терминах `not` и `or`, а оператор `or` может быть определен в терминах `not` и `and`. Однако, `and` и `or` настолько полезны, что во всех языках программирования есть и то, и другое.

Примечание 4. Встроенные типы данных на английском языке.


Python - Data Types
Numeric
Dictionary
Boolean
Set
Sequence Type
Integer
Complex
Float
Strings
List
Tuple

Примечание 5. Приведем таблицы истинности для логических операторов **`and`**, **`or`**, **`not`**:

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

a	not a
False	True
True	False

Примечание 6. Посмотреть фильм о Джордже Буле на русском языке можно по ссылке.

Функция bool()

Для приведения других типов данных к булеву существует функция bool(), работающая по следующим соглашениям:

- **строки:** пустая строка — ложь (False), непустая строка — истина (True);
- **числа:** нулевое число — ложь (False), ненулевое число (в том числе и меньшее нуля) — истина (True);
- **списки:** пустой список — ложь (False), непустой — истина (True).

Следующий программный код:

```
print(bool('Beegeek'))
print(bool(17))
print(bool(['apple', 'cherry']))
print(bool())
print(bool(''))
print(bool(0))
print(bool([]))
```

выведет:

```
True
True
True
False
False
False
False
```

Если функцию `bool()` вызвать без аргументов, то она вернет значение `False`.

Функции, возвращающие булево значение

Мы можем создавать функции, возвращающие булевы значения (`True` или `False`). Такая практика очень полезна. Напишем функцию `is_even()`, принимающую одно число и возвращающую значение `True`, если переданное число четное и `False` - в противном случае:

```
def is_even(num):
    return num % 2 == 0
```

Следующий программный код:

```
print(is_even(8))
print(is_even(7))
```

выведет:

```
True
False
```

В программировании функция, которая возвращает значение `True` или `False`, называется **предикатом**.

Функция isinstance()

В языке Python имеется встроенная функция **`isinstance()`** для проверки соответствия типа объекта какому-либо типу данных.

Следующий программный код:

```
print(isinstance(3, int))
print(isinstance(3.5, float))
print(isinstance('Beegeek', str))
print(isinstance([1, 2, 3], list))
print(isinstance(True, bool))
```

выведет:

```
True
True
True
True
True
```

Число 3 – целое число, число 3.5 – вещественное число, 'Beegeek' – строка и т.д.

Следующий программный код:

```
print(isinstance(3.5, int))
print(isinstance('Beegeek', float))
```

выведет:

```
False
False
```

так как число 3.5 не целое (`float` не `int`), 'Beegeek' - строка, а не вещественное число (`str` не `float`).

Функция type()

В языке Python имеется встроенная функция **`type()`**, позволяющая получить тип указанного в качестве аргумента объекта.

Следующий программный код:

```
print(type(3))
print(type(3.5))
print(type('Beegeek'))
print(type([1, 2, 3]))
print(type(True))
```

выведет:

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'bool'>
```

Функция `type()` часто бывает полезна при отладке программного кода, а также в реальном коде, особенно в объектно-ориентированном программировании с наследованием и пользовательскими строковыми представлениями, но об этом позже.

Обратите внимание, что при проверке типов обычно вместо функции `type()` используется функция `isinstance()`, так как она принимает во внимание **иерархию типов (ООП)**.

Тип данных NoneType

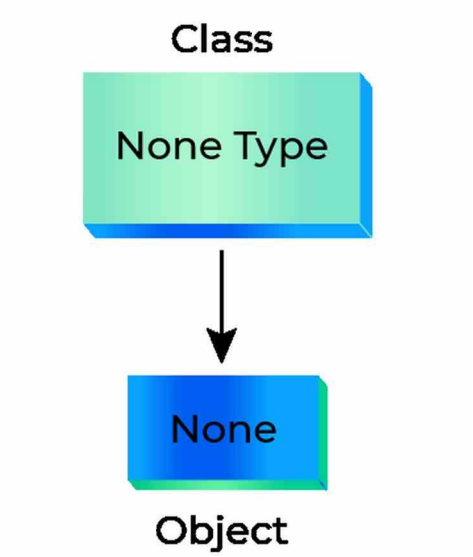
Пустое значение

Во многих языках программирования (Java, C++, C#, JavaScript и т.д.) существует ключевое слово `null`, которое можно присвоить переменным. Концепция ключевого слова `null` заключается в том, что оно дает переменной нейтральное или "нулевое" поведение.

В языке Python, слово `null` заменено на `None`, поскольку слово `null` звучит не очень дружелюбно, а `None` относится именно к требуемой функциональности – это ничего и не имеет поведения.

Литерал None

Литерал `None` в Python позволяет представить `null` переменную, то есть переменную, которая не содержит какого-либо значения. Другими словами, `None` – это специальная константа, означающая пустоту. Если более точно, то `None` – это объект специального типа данных `NoneType`.



Следующий программный код:

```
var = None
print(type(var))
```

выведет:

```
<class 'NoneType'>
```

Мы можем присвоить значение `None` любой переменной, однако мы не можем самостоятельно создать другой `NoneType` объект.

Все переменные, которым присвоено значение `None`, ссылаются на один и тот же объект типа `NoneType`. Создание собственных экземпляров типа `NoneType` недопустимо. Объекты, существующие в единственном экземпляре, называются синглтонами.

Проверка на None

Для того чтобы проверить значение переменной на `None`, мы используем либо оператор `is`, либо оператор проверки на равенство `==`.

Следующий программный код:

```
var = None
if var is None:  # используем оператор is
  print('None')
else:
  print('Not None')
```

выведет:

```
None
```

Следующий программный код:

```
var = None
if var == None:  # используем оператор ==
  print('None')
else:
  print('Not None')
```

выведет:

```
None
```

Для сравнения переменной с `None` всегда используйте оператор `is`. Для встроенных типов поведение `is` и `==` абсолютно одинаково, однако с пользовательскими типами могут возникнуть проблемы, так как в Python есть возможность переопределения операторов сравнения в пользовательских типах.

Сравнение None с другими типами данных

Сравнение `None` с любым объектом, отличным от `None`, дает значение `False`.

Следующий программный код:

```
print(None == None)
```

выведет:

```
True
```

Следующий программный код:

```
print(None == 17)
print(None == 3.14)
print(None == True)
```

```
print(None == [1, 2, 3])
print(None == 'Beegeek')
```

выведет:

```
False
False
False
False
False
```

Важно понимать, что следующий программный код:

```
print(None == 0)
print(None == False)
print(None == '')
```

выведет:

```
False
False
False
```

Значение `None` **не отождествляется** со значениями `0`, `False`, `''`.

Сравнивать `None` с другими типами данных можно только на равенство.

Следующий программный код:

```
print(None > 0)
print(None <= False)
```

приводит к ошибке:

```
TypeError: '>' not supported between instances of 'NoneType' and 'int' ('bool')
```

Примечания

Примечание 1. Обратите внимание, что функции, не возвращающие значений, на самом деле, в Python возвращают значение `None`:

```
def print_message():
    print('Я - Тимур,')
    print('король матана.')
```

Мы можем вызвать функцию `print_message()` так:

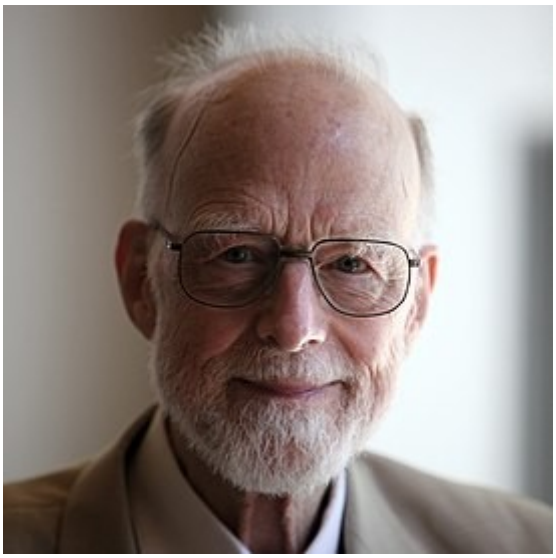
```
print_message()
```

или так:

```
res = print_message()
```

В переменной `res` хранится значение `None`.

Примечание 2. Концепция `null` значений появилась при создании языка `ALGOL W` великим Чарльзом Хоаром, который позднее назвал собственную идею ошибкой на миллиард долларов. Подробнее можно почитать тут.



Чарльз Хоар - автор одного из самых быстрых алгоритмов сортировки, основанной на сравнениях: быстрая сортировка (QuickSort).