Python akademie - lekce 2 - DD.MM.YYYY

# 02_01: Boolean

---

### Zajímavé odkazy z této lekce:

- Oficiální dokumentace boolean operátorů
- Oficiální dokumentace k ověřování členství
- Oficiální dokumentace porovnávacích operátorů
- Real Python
- Encoding Guide

---

## Boolean

---

Tento **datový typ** pomáhá rozhodovat, jestli je celý **zápis** či **hodnota** pravdivá a nebo nepravdivá.

Setkáme se s hodnotami:

- `True` (~pravda),
- `False` (~nepravda).

**Boolean** vychází z datového typu **integer** (~z celých čísel):

- `True` odpovídá hodnotě `1`,
- `False` odpovídá hodnotě `0`.

# Funkce `bool`

---

Tato zabudovaná funkce nám pomůže lépe pochopit práci s datovým typem **boolean**.

Funguje jako nějaký Python "rozhodčí", kterému stačí sdělit **konkrétní hodnotu** nebo **zápis** a ona ji vyhodnotí jestli je pravdivá či nikoliv.

```
bool(True)
```

```
bool(1)
```

```
bool(False)
```

```
bool(0)
```

```
bool(2)
```

```
probiha_akademie = True
```

```
print(probiha_akademie)
```

```
bool(probiha_akademie)
```

```
probiha_akademie2 = 1
```

```
bool(probiha_akademie2)
```

```
print(probiha_akademie2)
```

Podívej se jak funkce bool pracuje s běžnými hodnotami:

```
bool(0)
```

```
bool("Matous")
```

```
bool("")
```

```
bool(" ")
```

```
bool([]) # co to znamena? prazny string? -> PRAZDNY LIST
```

▶ ❓Klikni zde pro vysvětlení

```
bool([" "])
```

```
len([''])
```

```
bool(None)
```

Hodnotu `None` v Pythonu používáme pro označení absence hodnoty (odkaz nemá hodnotu). V jiných jazycích třeba `Null`.

Můžeme tedy prohlásit, že pokud vložím **neprázdnou hodnotu** (`str`, `int`, `list`), dostaneme hodnotu `True`. V opačném případě dostaneme `False`.

## Srovnávací operátory

| Operátor | Význam |
|---|---|
| `<` | menší než |
| `>` | větší než |
| `<=` | menší nebo rovno |
| `>=` | větší nebo rovno |
| `==` | rovnost |
| `!=` | nerovnost |
| `is` | totožná objektová identita |
| `is not` | různá objektová identita |

Nejprve **klasické srovnávací operátory**, které již známe:

```
bool(1 < 2)  # v interaktnivnim interpretu/notebooku lze pouzit bez funkce 'bool'
```

```
bool(100 > 2000)
```

```
bool(100 >= 99)
```

```
bool(5 == 5)
```

```
bool(5 != 5)
```

```
bool(True == False)
```

**Srovnání dvou hodnot** pomocí `==` a `!=`:

```
bool(10 == 10)
```

```
bool(5 == 10)
```

```
bool(5 != 10)
```

```
bool(10 != 10)
```

## Srovnávání identit?

---

Obecně platí, že všechno v Pythonu je **objekt**.

Každý objekt má svůj **identifikátor** (jako mají auta unikátní poznávací značky).

Tento **identifikátor** můžeš vypsat pomocí zabudované funkce id.

```
jmeno_1 = "Radim"
print(id(jmeno_1))
```

```
jmeno_2 = "Lukas"
print(id(jmeno_2))
```

```
prijmeni_1 = "Jedlicka"
print(id(prijmeni_1))
```

```
prijmeni_2 = "Jedlicka"
print(id(prijmeni_2))
```

Vidíme, že identifikátorem bude delší celé číslo. Pokud jsou **čísla stejná**, jde o **identickou hodnotu** v paměti našeho počítače.

```
bool(jmeno_1 is jmeno_2)
```

```
bool(jmeno_1 == jmeno_2)
```

```
prijmeni_1 is prijmeni_2
```

## Logické operace

---

Tyto operace jsou spojené s **boolean operátory**:

1. `and`,
2. `or`,
3. `not`.

Tyto operátory slouží ke spojování (kombinování) několika **boolean** hodnot.

### Operátor `and`

```
bool(True and True)
```

```
bool(True and False)
```

```
bool(False and True)
```

```
bool(False and False)
```

```
bool(True and True and True)
```

```
bool(True and True and True and False)
```

Pokud použijeme **boolean operátor** `and` a od jednotlivých výroků získám byť jednu hodnotu `False`, **celý výsledek** bude **nepravdivý**. Tedy `False`.

Často můžeme aplikovat princip tzv. *zkráceného vyhodnocování*. Pokud uvidím jako první hodnotu `False` (a pracuji s operátorem `and`), nemusíš procházet další výrazy. Výsledek je `False`.

## Operátor `or`

```
bool(True or True)
```

```
bool(True or False)
```

```
bool(False or True)
```

```
bool(False or False)
```

```
bool(False or False or False)
```

```
bool(False or False or True)
```

Pokud použijeme **boolean operátor** `or` a od jednotlivých výroků získám byť jednu hodnotu `True`, **celý výsledek** bude **pravdivý**. Tedy `True`.

Opět lze použít princip **zrychleného vyhodnocování**. Tedy pokud funkce `bool` vrátí aspoň jednu hodnotu `True` (a pracuji s operátorem `or`), výsledek celého zápisu bude `True`.

## Operátor `not`

```
bool(not True)
```

```
bool(not False)
```

Jde o operaci, kterou lze aplikovat na **jedinou boolean** hodnotu, kdy výsledkem je její obrácená hodnota.

## Ověření členství (~membership testing)

---

Nejde o proces, který by přímo pracoval s **boolean** hodnotami.

Ovšem právě `True` a `False` jsou hodnoty, které jsou výsledkem toho procesu.

V podstatě jde o jednoduché zjišťování, jestli je konkrétní prvek členem konkrétní sekvence (obecně platí pro `str`, `list` a `tuple`).

```
bool("m" in "Matous")
```

```
bool("M" in "Matous")
```

```
bool("@" not in "Matous")
```

```
bool("a" not in "Matous")
```

Na levé straně od `in` vidíme string `"m"` a na pravé straně `"Matous"`.

V podstatě se ptáme následovně:"**Je výraz na levé straně součásti výrazu na pravé straně?**".

Pokud je odpovědí ano, potom Python vypíše hodnotu `True`. V opačném případě `False`.

**Ověření členství** je proces, který obecně zařazujeme mezi operace jako *indexing*, *slicing*, *striding*. Resp. **operace, které můžeme provádět se sekvencemi**.

### `is` nebo `==` ?

Opatrně na aplikaci operátorů porovnávání **identit** a **hodnot**:

- `==` a `!=` porovnávají hodnotu dvou objektů
- `is` a `is not` porovnávají, jestli dvě proměnné ukazují v paměti počítače na stejný objekt

```
muj_seznam_1 = [1, 2, 3]
muj_seznam_2 = [1, 2]
```

```
muj_seznam_1 == muj_seznam_2  # [1, 2, 3] == [1, 2]
```

```
muj_seznam_3 = [1, 2, 3]
muj_seznam_4 = [1, 2, 3]
```

```
muj_seznam_3 == muj_seznam_4
```

```
id(muj_seznam_3)
```

```
id(muj_seznam_4)
```

```
muj_seznam_3 is muj_seznam_4
```

▶ ❓ Klikni zde pro vysvětlení

## 😉 Zrádné syntaxe

---

```
True == 1
```

```
True + 11
```

```
["Matous", "Marek"][True]
```

```
print(["Matous", "Marek"][False])
```

```
int(True)
```

```
1 < 5 < 10  # 1 < 5 a 5 < 10
```

```
'1' == 1
```

```
10 == 10.0
```

```
["Matous", "Lukas"] == ["Matous", "Lukas"]
```

```
["Matous", "Lukas"] == ["Lukas", "Matous"]
```

```
id([])           # proc jsou tyto dva idy rozdilne???
```

```
id([])
```

```
[] is []
```

```
bool("a" < "d")
```

```
ord("A")
```

```
ord("d")
```

```
ord("@")
```

```
chr(64)
```

```
chr(100)
```

```
chr(65)
```