



OSNOVA

&lt; PŘEHLED KURZU

100% hotovo z Lekce 2

Vyhledávat



# ÚVOD DO DRUHÉ LEKCE

## Obsah

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / ÚVOD DO DRUHÉ LEKCE / OBSAH

V této lekci nás čeká následující:

1. **Instalace Pythonu** na tvém počítači,
2. **instalace vhodného pracovního prostředí** (doporučujeme PyCharm),
3. datový typ **boolean**,
4. **podmínkový zápis**,
5. **metody** v Pythonu,

## Instalace Pythonu

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / ÚVOD DO DRUHÉ LEKCE / INSTALACE PYTHONU

Instalace Pythonu na Windows 10 můžeš omrknout ve videu níže.



Vyhledávejte na webu, aniž byste byli sledováni



**Už vás nebabí, že jste sledováni online? Rádi vám s tím pomůžeme.**

Získejte bezproblémovou ochranu osobních údajů ve svém prohlížeči zdarma v jediném stahování:

Soukromé Vyhledávání  Blokování Trackerů  Šifrování Stránek

Přidat DuckDuckGo do Chrome

★★★★★ Hodnoceno 4.4/5



Odkaz pro stažení instalovačního souboru **pro Windows** najdeš [zde](#).

## Linux

Novější verze Linuxů už **mají dostupný Python** i ve verzi 3+. Přesto si v příkazovém rádku ověř, jestli je to pravda:

```
python3 -V
```

Pokud máš pouze nižší verze, tady je postup instalace Pythonu pro distribuce **Debian**, **Ubuntu**, **Mint**.

Nejprve potřebné aktualizace a balíčky:

```
sudo apt update  
sudo apt-get install software-properties-common
```

Přidáme repozitář s Pythonem, aktualizujeme jej a nainstalujeme jazyk:

```
sudo add-apt-repository ppa:deadsnakes/ppa  
sudo apt-get update  
sudo apt install python3.9
```

Po instalaci ověř, jestli máš aktivní správnou verzi:

```
python3.9 -V  
# výstup by měl vypadat: Python 3.9.0
```

## MacOS

MacOs stejně jako Linuxy má Python již zabudovaný. Ověř v příkazovém řádku jestli příkaz vrátí správnou verzi:

```
python3 -V
```

Pokud bude výsledná hláška chybová, přejdi na [oficiální web](#) a vyber v tabulce dole stažení instalačního souboru.

## Výběr pracovního prostředí

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / ÚVOD DO DRUHÉ LEKCE / VÝBĚR PRACOVNÍHO PROSTŘEDÍ

**Pracovní prostředí** je nějaký zápisník, ve kterém budeš zapisovat instrukce pro interpret Pythonu, který je po spuštění provede.

Teoreticky by ti stačil jakýkoliv **textový editor** jako je *poznámkový blok*, ale jsou vhodnější nástroje, které podporují formátování zápisu, zvýrazňování nedostatků, konzistentní odsazování, aj.

### Možnosti při výběru editorů aj.

**Můžeš si vybrat jednu z těchto možností** (jsou zde uvedené pouze některé, výběr je daleko větší):

1. **Interaktivní interpret** - interaktivní prostředí interpreta přímo v příkazovém řádku tvého počítače. Takto můžeš okamžitě spouštět a ověřovat jednoduché příkazy. (objeví černá obrazovka se třemi šipkami na začátku posledního řádku **>>>**).
2. **Textový editor** - upravený textový editor, který umí formátovat text, napovídат, atd. Na ukázku se můžeš podívat třeba na [Sublime text](#) nebo [Atom](#).

3. **Vývojařské prostředí** - specializované prostředí obsahující různé nástroje a pomůcky (sofistikovanější, často větší než editor i náročnější na paměť). Podívej se třeba na [PyCharm](#), [Visual Studio Code](#) nebo webové prostředí [Replit](#).

4. **Notebooky** - speciální prostředí, které umožňuje využít potenciál interpretu a současně zapisovat poznámky, zobrazovat obrázky, grafy aj. Přečíst si můžeš o [projektu Jupyter](#) nebo [Google Colaboratory](#).



## PyCharm

Instrukce pro instalaci a první spuštění najdeš na videu níže. Případně odkaz na oficiální web omrkni [zde](#).



# BOOLEAN

## Úvod

Jméno datového typu **boolean** (případně *bool*) možná čteš poprvé. Určitě ti ale nejsou cizí výrazy **True** a **False** (**~pravda** a **nepravda**). Právě tyto výrazy jsou **hodnotami** datového typu **boolean**.

Podstatou tohoto datového typu je **vyhodnocení, jestli je zápis (nebo jen hodnota) pravdivý nebo nepravdivý**.

Vše bude nejlepší popsat na krátké úvodní ukázce:

```
>>> 6 > 4  
True  
>>> 1000 < 100  
False
```

V prvním páru srovnáváme jestli je číslo **6** větší než číslo **4**. Dobре víme, že takový zápis je **pravdivý**. Python to ví také! Proto je zápis ohodnocený *boolean hodnotou* **True**.

Ve druhém páru porovnáváme, jestli je číslo **1000** menší než číslo **100**. Logika věci nám říká, že to **není pravda**. Přesně tak postupuje i Python (tedy jeho interpret), proto vidíme výslednou hodnotu **False**.

Jak je ale možné, že Python automaticky rozumí našemu zápisu s čísly a srovnávacími operátory? Pokud ti tato otázka vrtá hlavou, pokračuj ve čtení materiálů.

## Jen pro zvídavé

Datový typ **boolean** je v podstatě podmnožinou datového typu **int** (tedy celých čísel). Hodnota **True** totiž vychází z celého čísla **1**, zatímco hodnota **False** vychází z čísla **0**.

```
>>> True == 1 # '==' operátor pro srovnání dvou hodnot  
True  
>>> False == 0  
True
```

## Funkce bool

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / BOOLEAN / FUNKCE BOOL

Abychom mohli více porozumět datovému typu **boolean**, je užitečné zmínit se o pomocné zabudované funkci **bool**.

Tato předpřipravená funkce pracuje pro představu jako **soudce**, který vynáší verdikt nad našimi zápisy.

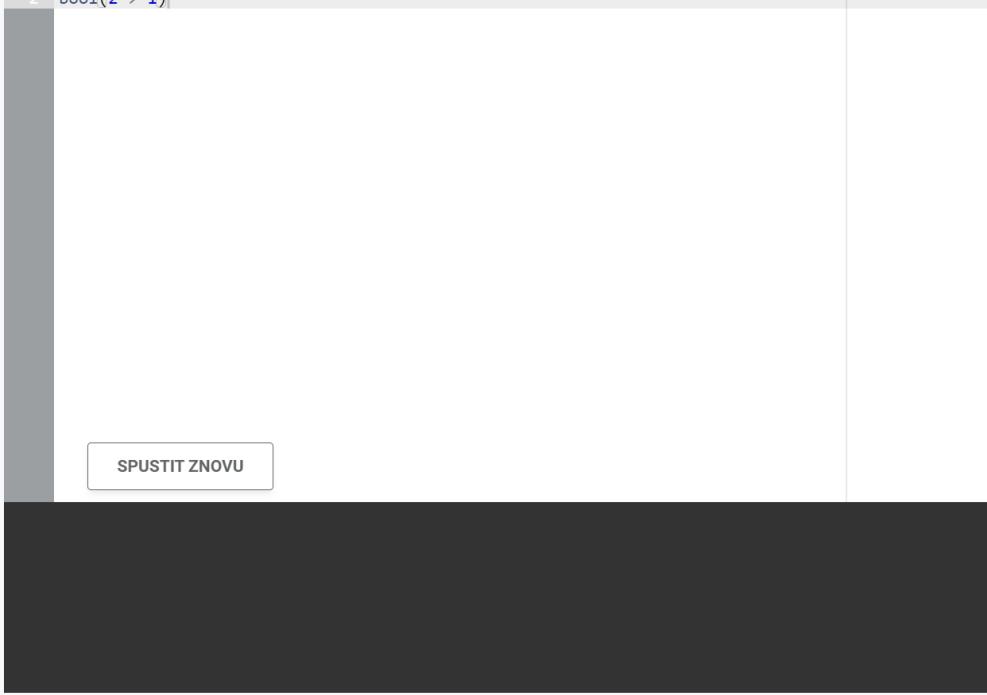
```
>>> bool(2 > 1)  
True  
>>> bool(5 != 1)  
True
```

Všimni si, že pokud funkci **bool** nezapíšeš v rámci **interaktivní prostředí interpretu**, vůbec mu to nevadí a vypíše ti stejný výstup!

```
>>> 2 > 1  
True  
>>> 5 != 1  
True
```

Takové chování je typické pro prostředí interpretu, ale **dávej pozor!** Pokud budeš zapisovat stejný zápis pro Python u sebe do souboru, vždycky použij jméno funkce.

```
1 # Prostředí podobné editoru na tvém počítači  
2 bool(2 > 1)
```



SPUSTIT ZNOVU

Pokud sis spustil zápis v našem **editoru** v předchozí ukázce a **nic se nestalo**, neboj, takhle je to totiž **v pořádku**.

Editor, kam si zapsal svůj příkaz, se totiž **chová jinak než interaktivní interpret** Pythonu. V editoru musíš zadávat příkazy explicitně. Takže pokud zapíšeš pouze `bool(1 < 2)`, Python tato čísla **porovná**, ale **boolean hodnotu** nevypíše, protože k tomu nedostal příkaz.

Aby se tento výstup zobrazil i tobě, musíš použít funkci `print`, skrze kterou umí Python vypisovat výstup.

```
1 # Prostředí podobné editoru na tvém počítači
2 print(bool(2 > 1))
```

SPUSTIT ZNOVU

True

## Srovnávací operátory

Právě **srovnávání** je první proces, ve kterém se s datovým typem **boolean** setkáš.

V Pythonu je k dispozici 8 různých **srovnávacích operátorů**. Jak se můžeš přesvědčit v tabulce níže:

Operace	Význam
<code>&gt;</code>	je větší
<code>&gt;=</code>	je větší nebo roven
<code>&lt;</code>	je menší
<code>&lt;=</code>	je menší nebo roven
<code>==</code>	je roven
<code>!=</code>	není roven
<code>is</code>	totožná objektová identita
<code>is not</code>	různá objektová identita

Pokud neznáš všechny, nic si z toho nedělej. V další sekci si jednotlivé srovnávací operátory popíšeme.

## Základní srovnávací operátory

**Základní srovnávací operátory** jsou poměrně rozšířené a intuitivní. Určitě ale nenech nic náhodě a vyzkoušej si jejich aplikaci:

```
>>> 10 > 1
True
>>> 10 > 10
False
>>> 10 >= 10
True
>>> 1 < 4
True
>>> 5 < 6
True
>>> 6 <= 6
True
```

## Srovnávání dvou hodnot

**Srovnávání hodnot** spočívá v použití dvou operátorů `==` (jsou si rovné) a `!=` (nejsou si rovné). Účelem může být třeba porovnání údajů uložených ve dvou různých proměnných:

```
>>> 10 == 11
False
>>> 11 == 11
True
>>> 4 != 5
True
>>> 5 != 5
False
```

## Porovnávání identit

**Srovnávání identit** může na první pohled vypadat komplikovaně, ale my tomu společně přijdeme na kloub.

Obecně platí, že **všechno v Pythonu je objekt**. Každá hodnota (1, 3.141), každá sekvence ([ "a", "b"] , (1, 2, 3)), zkrátka všechno.

Každý takový výraz má nějaké svoje **označení**. Jako mají třeba auta různé poznávací značky. Takové označení můžeš zjistit pokud použiješ zabudovanou funkci **id**.

```
>>> id(1)
9784896
>>> id("a")
140462351325488
>>> moje_cislo = 2
>>> id(moje_cislo)
9784928
```

Funkce **id** v podstatě udělá jen to, že vrátí celé číslo (tedy označení identity) objektu.

Takže pokud budeš chtít zkontovalovat, **jestli jsou dvě hodnoty identické** (pro zvídavé, jestli mají stejné číslo, tedy adresu objektu v paměti počítače) napíšeš následující:

```
>>> cislo_1 = 1
>>> cislo_2 = 2
```

Vytvoříš dvě proměnné, každá má jinou hodnotu.

```
>>> id(cislo_1)
9784896
>>> id(cislo_2)
9784928
```

Ověříš si, že funkce **id** vrací různé čísla.

```
>>> cislo_1 is cislo_2
False
```

Potom i operátor **is** ukáže, že jde o dva různé objekty.

Můžeš ověřit i dvě stejné hodnoty:

```
>>> cislo_3 = 5
>>> cislo_4 = 5
>>> id(cislo_3)
9785024
>>> id(cislo_4)
9785024
>>> cislo_3 is cislo_4
True
```

Nyní vidíš, že funkce **id** vrací stejná čísla. Takže i **boolean hodnota** bude pravdivá.

## Logické operace

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / BOOLEAN / LOGICKÉ OPERACE

Jednoduché **boolean** výrazy následně můžeš spojovat pomocí **boolean operátorů** a provést s nimi různé logické operace.

Mezi **boolean operátory** patří:

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

```
>>> True and True  
True  
>>> False and True  
False
```

Jak ale tyto operátory pracují? Proč mi `and` vrací jednou `True` a podruhé `False`? Odpověď najdeš v tabulkách níže.

---

## Logický operátor and

Tento *boolean* operátor můžeš používat nejen na dva výrazy, ale na celou řadu výrazů (tedy 3 a více).

Operace	Výsledek
<code>True and True</code>	<code>True</code>
<code>True and False</code>	<code>False</code>
<code>False and True</code>	<code>False</code>
<code>False and False</code>	<code>False</code>
<code>True and True and True</code>	<code>True</code>
<code>True and True and False</code>	<code>False</code>

Při pohledu na tabulku výš můžeme říct, že pokud budeme mít výrazy spojené **boolean operátorem and**, bude výsledek `True` jenom tehdy, **pokud všechny výrazy budou mít hodnotu True**.

Současně tu platí tzv. *zkrácené vyhodnocení*, tzn. že pokud Python prochází naše výrazy (spojené `and`) a uvidí výraz `False`, ani se na ostatní výrazy nepodívá a vrací `False`.

---

## Logický operátor or

Stejně jako `and`, boolean operátor `or` můžeš používat na dva výrazy a více.

Operace	Výsledek
<code>True or True</code>	<code>True</code>
<code>True or False</code>	<code>True</code>
<code>False or True</code>	<code>True</code>
<code>False or False</code>	<code>False</code>
<code>False or False or True</code>	<code>True</code>
<code>False or False or False</code>	<code>False</code>

Pokud budeme mít výrazy spojené **boolean operátorem or**, bude výsledek `True` tehdy, **pokud alespoň jeden výraz bude mít hodnotu True**.

Dále i tu platí *zkrácené vyhodnocení*. Takže pokud Python prochází naše výrazy (tentokrát spojené `or`) a uvidí výraz `True`, ani se na ostatní výrazy nepodívá a vrací `True`.

---

## Logický operátor not

Boolean operátor `not` se používá pouze na jednu **boolean hodnotu** a vypíše její logický opak.

Operace	Výsledek
<code>not True</code>	<code>False</code>
<code>not False</code>	<code>True</code>

Pokud budeš mít mezi výrazy všechno logické operátory, dbej na **správné pořadí**, ve kterém budeš jednotlivé operátory vyhodnocovat:

1. Nejprve `not`,
2. následně `and`,
3. nakonec `or`.

## Ověřování členství

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / BOOLEAN / OVĚŘOVÁNÍ ČLENSTVÍ

Ačkoliv tato operace **nepatří** přímo k datovému typu **boolean**, přesto stojí za to o ní informovat právě nyní.

Výsledek **ověřování členství** (~membership testing) je totiž právě `True` nebo `False`, tedy **boolean hodnota**.

Opět si všechno ukážeme na krátké ukázce:

```
>>> "M" in "Matous"
True
>>> "@" not in "Matous"
True
>>> "@" in "Matous"
False
```

Na levé straně od `in` vidíme string `"M"` a na pravé straně `"Matous"`. V podstatě se ptáme následovně: "**Je výraz na levé straně součástí výrazu na pravé straně?**".

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

## Jen pro zvídavé

Ve skutečnosti patří **ověřování členství** (~membership testing) do stejné skupiny jako indexování, slicing, striding, aj. Tedy do skupiny *společných operací pro sekvenční datové typy*.

Všechny operace pro sekvenční datové typy najdeš v tabulce níž:

Operace	Výsledek	Název operace
<code>"a" in "auto"</code>	<code>True</code>	membership testing (ověření členství)
<code>"a" not in "auto"</code>	<code>False</code>	membership testing (ověření členství)
<code>"a" + "b"</code>	<code>"ab"</code>	concatenation (spojování)
<code>"a" * 3</code>	<code>"aaa"</code>	repetition (opakování)
<code>[1, 11, 111][0]</code>	<code>1</code>	indexing (indexování)
<code>[1, 11, 111][1:3]</code>	<code>[11, 111]</code>	slicing (rozkrájení sekvence)
<code>[1, 11, 111][0:3:2]</code>	<code>[1, 111]</code>	striding (přeskakování v sekvenci)

## Kvíz

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / BOOLEAN / KVÍZ

Vyber **boolean** hodnotu

A. **list**

B. **str**

C. **True**

D. **print**

E. **append**

## PODMÍNKOVÝ ZÁPIS

### Rozhodování

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / ROZHODOVÁNÍ

Co když budeš potřebovat, aby se tvůj zápis uměl sám **rozhodovat**?

Na základě dostupných hodnot uměl **rozhodnout**, jestli je např. uživatel starší 18 let, jestli proměnná obsahuje čísla, nebo jenom text.

V Pythonu tohoto **rozhodování** snadno dosáhneš právě pomocí **boolean hodnot**. Tedy pokud bude tebou zadáný výraz pravdivý či nikoliv, můžeš specifikovat dva různé průběhy tvého programu.

### Předpis podmínky if

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / PŘEDPIS PODMÍNKY IF

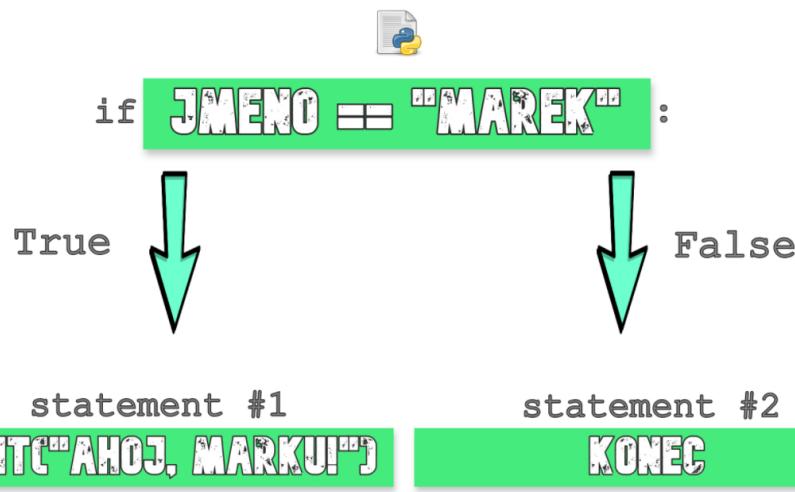
Pokud budeš potřebovat zapsat podobný **rozhodovací proces**, budeš potřebovat tzv. **podmínkový zápis**.

Takový krátký **podmínkový zápis** si můžeš prohlédnout níž:

```
1 # Vyzkoušej si spustit tento zápis!
2 jméno = "Marek"
3
4 if jméno == "Marek":
5     print("Ahoj, Marku!")
```

SPUSTIT ZNOVU

Ahoj, Marku!



Na na výše zmíněném příkladu si všimni těchto **charakteristických rysů** u podmínkové zápisu:

1. `if`, na začátku předpisu oznamuje, že se jedná o předpis podmínky,
2. `jmeno == "Marek"`, samotná podmínka, která je následně vyhodnocená `True` / `False`,
3. `:`, na konci předpisu nesmí chybět dvojtečka ukončující rádek,
4. **odsazení**, další rádek **musí** být odsazený o **4 mezery**, nebo **1 tabulátor**.

Pojď se nyní podívat jak takový podmínkový zápis prochází interpret Pythonu: ~ Nyní si rozebereme postup, který interpret Pythonu provádí pokaždé, pokud uvidí předpis podmínkového zápisu:

1. Na řádku s `if` vyzkouší zápis, který chceme ověřit (pro představu, na ověření můžeš použít funkci `bool(jmeno == "Marek")`),
2. pokud funkce `bool` vrátí hodnotu `True`, provede odsazený zápis, `print("Ahoj, Marku!")`
3. pokud funkce `bool` vrátí hodnotu `False`, přeskočí odsazený zápis,
4. pokračuje v dalším zápisu, pokud žádný není, skončí.

```
1 # Vyzkoušej si spustit tento zápis!
2 jmeno = "Lukas"
3
4 if jmeno == "Marek":
5     print("Ahoj, Marku!")
```

[SPUSTIT ZNOVU](#)

Pokud je vyzkoušený zápis **nepравdivý**, tedy `False`, můžeš si všimnout, že se nic nestane.

Právě proto, že je zápis **nepравdivý**, dojde k **přeskočení odsazeného zápisu** a Python pokračuje dalším zápisem. Tady ale žádný další zápis není, a proto se ukončí (resp. nic nevypíše).

## Předpis if/else

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / PŘEDPIS IF/ELSE

Velice často se dostaneš do situace, kdy budeš od tvého zápisu požadovat, aby fungoval *bud'*jedním způsobem, *nebo* druhým způsobem.

Budeš tedy potřebovat **dva různé průběhy** (*~control-flow*). Někdy také říkáme, že náš podmínkový zápis bude mít **dvě podmínkové větve**:

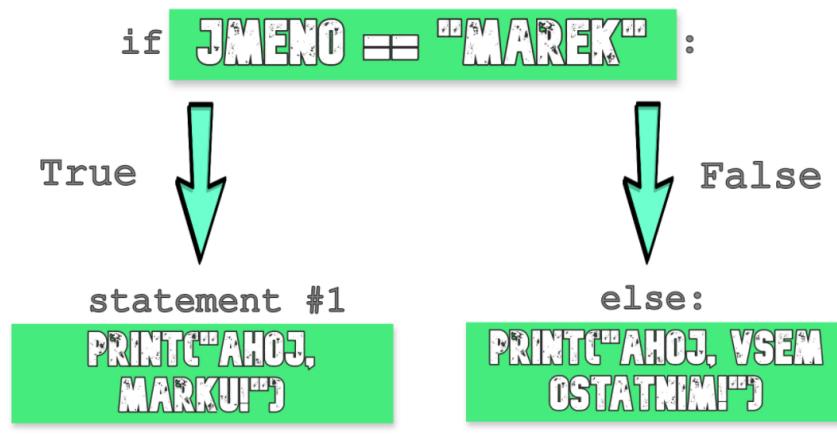
1. **Bud'** proved' toto ohlášení (větev začíná s klíčovým slovem `if`),
2. **nebo** proved' toto ohlášení (větev začíná s klíčovým slovem `else`).

```
1 # Vyzkoušej si spustit tento zápis!
2 jmeno = "Lukas"
3
4 if jmeno == "Marek":
5     print("Ahoj, Marku!")
6 else:
7     print("Ahoj, vsem ostatním!")
```

[SPUSTIT ZNOVU](#)

Ahoj, vsem ostatním!





Rozbor předchozího příkladu vychází z podobných kroků, jako náš úvodní zápis se samotným `if`:

1. Na řádku s `if` vyzkouší zápis, který chceme ověřit (pro představu, na ověření můžeš použít funkci `bool(jmeno == "Marek")`).
2. pokud funkce `bool` vrátí hodnotu `True`, provede odsazený zápis, `print("Ahoj, Marku!")`
3. pokud funkce `bool` vrátí hodnotu `False`, provede zápis ve větvi `else`, tedy `print("Ahoj, vsem ostatnim!")`
4. pokračuje v dalším zápisu, pokud žádný není, skončí.

Současně si všimni, že ohlášení zapsané ve větvi `else` je odsazené stejným způsobem jako u větve `if`.

Vyzkoušej si v ukázce výš přepsat výraz v proměnné `jmeno = "Marek"`. Potom spusť celou ukázku ještě jednou a sleduj nový výstup.

## Předpis if/elif/else

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / PŘEDPIS IF/ELIF/ELSE

Může ovšem nastat situace, kdy budeš potřebovat 3 a více různých podmínkových větví.

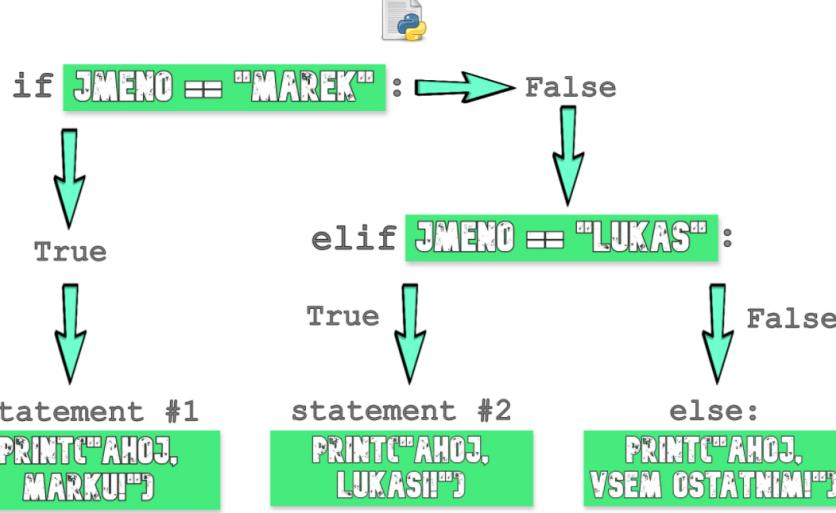
V takovém případě potřebuješ vytvořit **minimálně 3 různé podmínkové větvě**:

1. **Bud'** proved' toto ohlášení (větev začíná s klíčovým slovem `if`),
2. **jinak** proved' toto ohlášení (větev začíná s klíčovým slovem `elif`),
3. **nebo** proved' toto ohlášení (větev začíná s klíčovým slovem `else`).

```

1 # Vyzkoušej si spustit tento zápis!
2 jmeno = "Lukas"
3
4 if jmeno == "Marek":
5     print("Ahoj, Marku!")
6 elif jmeno == "Lukas":
7     print("Ahoj, Lukasi!")
8 else:
9     print("Ahoj, vsem ostatnim!")

```



Rozbor předchozího příkladu bude vypadat následovně:

1. Na řádku s `if` vyzkouší zápis, který chceme ověřit (pro představu, na ověření můžeš použít funkci `bool(jmeno == "Marek")`),
2. pokud je podmínka u `if` vyhodnocená jako `True`, provede odsazené ohlášení pod `if` a přeskočí další větve,
3. pokud je podmínka u `if` vyhodnocená jako `False`, pokračuje k podmínce v `elif`,
4. pokud je podmínka u `elif` vyhodnocená jako `True`, provede odsazené ohlášení pod `elif` a přeskočí další větve,
5. pokud je podmínka u `elif` vyhodnocená jako `False`, provede odsazené ohlášení v `else`,
6. pokračuje v další zápisu, pokud žádný není, skončí.

Znovu platí, že ohlášení zapsané ve věti `elif` je odsazené stejným způsobem jako u větev `if` a `else`.

Zatím co v jednom podmínkovém zápisu (stromu) můžeš mít jednu větev `if` a jednu větev `else`, podmínkových větví `elif` můžeš mít kolik chceš.

Vyzkoušej si v ukázce výš přepsat výraz v proměnné `jmeno = "Marek"`, `jmeno = "Matous"`.

## Doplňení boolean operátory

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / DOPLNĚNÍ BOOLEAN OPERÁTORY

Pokud budeš potřebovat zohlednit více výrazů v rámci jedné podmínkové větve, můžeš využít logických operací `and`, `or` a `not`.

Podívejme se společně na další ukázku:

```
1 jmeno = "Marek"
2 vek = 20
3
4 if jmeno == "Marek" and vek >= 18:
```

```
1 # jmeno -- Marek a vek >= 18.
2     print("Bezva, Marku, muzes s nami na pivo.")
3 else:
4     print("Bohuzel, jeste nemuzes pit.")
```

V předpisu větve `if` vidíme dvě podmínky, které je nutné vyhodnotit:

1. `bool(jmeno == "Marek")`,
2. `bool(vek >= 18)`.

Tyto dvě ohlášení jsou spojená pomocí **boolean operátoru and**. Tedy abychom dostali `True` z první podmínkové větve (`if`), je potřeba, aby obě podmínky (číslo 1 a 2) byly pravdivé (`True`).

```
1 # Vyzkoušej si spustit tento zápis!
2 # Nasledne zkus zmenit hodnoty v promennych 'jmeno' a 'vek'
3 jmeno = "Marek"
4 vek = 20
5
6 if jmeno == "Marek" and vek >= 18:
7     print("Bezva, Marku, muzes s nami na pivo.")
8 else:
9     print("Bohuzel, jeste nemuzes pit.")
```

**SPUSTIT ZNOVU**

Bezva, Marku, muzes s nami na pivo.

Podobně můžeš pracovat i s dalšími **boolean operátory** a spojovat několik podmínek v rámci jedné větve (`if` nebo `elif`).

## Ternární operátor

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / TERNÁRNÍ OPERÁTOR

### Jen pro zvídavé

Pokud ti přišel **standartní zápis** podmínek jednoduchý, předvedeme si další způsob, jakým někdy lze napsat podmínku typu **bud/nebo**.

Obecně vypadá zápis takhle:

```
1 X if Y else Z
```

1. Proveď ohlášení `X`,
2. pokud je podmínka `Y` pravdivá,
3. jinak proveď ohlášení `Z`.

Celý podmínkový strom, který by se běžně skládal z větví `if` / `else`, je nyní zapsaný pomocí **jediného rádku**.

```
1 # Vyzkoušej si spustit tento zápis!
2 jmeno = "Marek"
```

```
3  
4 print("Ahoj, Marku!") if jmeno == "Marek" else print("Ahoj, vsem!")
```

SPUSTIT ZNOVU

Ahoj, Marku!

Pokud ti ale tato varianta nepřijde dostatečně **názorná** a **čitelná**, určitě můžeš nadále používat **klasický podmínkový zápis**.

## Kvíz

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / PODMÍNKOVÝ ZÁPIS / KVÍZ

1/4

Co musí obsahovat jednoduchý podmínkový předpis (pouze s **if** větví)?

A. **if**, podmínku/podmínky, **;**, odsazené ohlášení

B. **if**, podmínku/podmínky, **,**, odsazené ohlášení

C. **if**, podmínku/podmínky, **:**, odsazené ohlášení

D. **if**, podmínku/podmínky, **,**, ohlášení

E. **if**, podmínku/podmínky, **;**, ohlášení

## METODY

# Úvod

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / METODY / ÚVOD

Kromě **zabudovaných funkcí** můžeme v Pythonu pracovat s tzv. *metodami datových typů*.

Metoda **je pomocný nástroj**, který nám umožňuje lepší zacházení a pracování s konkrétními datovými typy.

## Rozdíl mezi funkcemi a metodami

Pro naše účely nám bude stačit toto rozdělení:

1. **Funkce** jsou obecnější, tzn. že mají širší použití pro různé datové typy (`print`),
2. **metody** jsou úzce zaměřené na konkrétní datový typ (`str`, `list`).

## Ukázky funkce

```
1 # Funkce 'print' umí pracovat s různými datovými typy
2 print(1)      # int
3 print(1.00)    # float
4 print("Jedna") # str
```

SPUSTIT ZNOVU

```
1
1.0
Jedna
```

## Ukázky metody

```
1 # Metody 'upper' a 'title' umí pracovat jen se stringem
2 print("matous".upper())  # funkce 'print' je potřeba kvůli vypsání výsledku
3 print("matous".title())
```

SPUSTIT ZNOVU

MATOUS  
Matous

## Spouštění metody

1. **Použití metody** se odvíjí od konkrétního výrazu (hodnoty) případně proměnné,
2. **s tečkou** spojíme hodnotu a jméno metody,
3. za metodou stojí kulatá závorka, která může a nemusí obsahovat dodatečné hodnoty.

## Metody stringů

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / METODY / METODY STRINGŮ

Metoda	Použití
<code>capitalize</code>	převede string na první znak velkým písmenem a zbytek malými písmeny
<code>casefold</code>	převede string na malá písmena
<code>center</code>	zarovná string na střed
<code>count</code>	vrací počet výskytu zadané hodnoty ve stringu
<code>encode</code>	vrací kódovaný string
<code>endswith</code>	vrací <code>True</code> pokud string končí zadanou hodnotou
<code>expandtabs</code>	nastaví délku tabulátoru stringu
<code>find</code>	hledá ve stringu zadanou hodnotu a vrací index, na kterém najde hodnotu
<code>format</code>	formátuje konkrétní hodnotu na string
<code>format_map</code>	formátuje konkrétní hodnotu na string
<code>index</code>	hledá ve stringu zadanou hodnotu a vrací index, na kterém najde hodnotu
<code>isalnum</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu alfanumerické
<code>isalpha</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu abecedické
<code>isdecimal</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu desetičíslo
<code>isdigit</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu číslo
<code>isidentifier</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu klíčový identifikátor Pythonu
<code>islower</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu malým písmem
<code>isnumeric</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu číselné znaky
<code>isprintable</code>	vrací <code>True</code> pokud lze všechny znaky ve stringu vypsat
<code>.....</code>	vrací <code>True</code> pokud jsou všechny znaky ve stringu

<code>isspace</code>	mezera
<code>istitle</code>	vrací <code>True</code> pokud string začíná velkým písmenem a následují malá písmena
<code>isupper</code>	vrací <code>True</code> pokud string obsahuje jen velká písmena
<code>join</code>	spojí sekvenci údajů pomocí zadané hodnoty
<code>ljust</code>	zarovná string doleva
<code>lower</code>	převede string na malé písmena
<code>lstrip</code>	ořízne zadaný string o specifikovaný symbol (na začátku a na konci) (vlevo)
<code>partition</code>	Returns a tuple where the string is parted into three parts
<code>replace</code>	v zadaném stringu nahradí konkrétní symboly, jinými symboly
<code>rfind</code>	hledá ve stringu zadanou hodnotu a vrací index, na kterém najde hodnotu (vpravo)
<code>rindex</code>	hledá ve stringu zadanou hodnotu a vrací index, na kterém najde hodnotu (vpravo)
<code>rjust</code>	zarovná string doprava
<code>rsplit</code>	rozdělí string pomocí zadанého znaku, jinak podle mezer a newlinů (vpravo)
<code>rstrip</code>	ořízne zadaný string o specifikovaný symbol (na začátku a na konci) (vpravo)
<code>split</code>	rozdělí string pomocí zadaného znaku, jinak podle mezer a newlinů (vlevo)
<code>splitlines</code>	rozdělí string pomocí newlinů
<code>startswith</code>	vrací <code>True</code> pokud string začíná zadanou hodnotou
<code>strip</code>	ořízne zadaný string o specifikovaný symbol (na začátku a na konci) (vlevo)
<code>swapcase</code>	převede malá písmena na velká a obrácené
<code>title</code>	převede string na první znak velkým písmenem a zbytek malými písmeny
<code>upper</code>	převede zadaný string na velká písmena
<code>zfill</code>	doplní string o nuly na levé straně v zadané délce

## Ukázky

```
1 # Metoda 'index' najde hledanou hodnotu ve stringu a vrátí její index
2 print("ukulele".index("k"))
```

SPUSTIT ZNOVU

```
1
```

```
1 # Metoda 'replace' najde první zadanou hodnotu ve stringu a nahradí ji druhou zadanou hodnotou
2 print("pi@no".replace("@", "a"))|
```

SPUSTIT ZNOVU

```
piano
```

```
1
```

```
1 # Metoda 'isupper' zkontroluje string a vypíše 'True' pokud jsou všechny znaky velkými písmeny
2 print("KYTARA".isupper())|
```

SPUSTIT ZNOVU

```
True
```

## Metody listů

Metoda	Použití
<code>append</code>	přidá údaj na konec listu
<code>clear</code>	odstraní všechny údaje z listu
<code>copy</code>	vrátí kopii listu
<code>count</code>	vrací počet výskytů zadané hodnoty v listu
<code>extend</code>	přidá údaj na konec listu
<code>index</code>	hledá v listu zadanou hodnotu a vrací index, na kterém najde hodnotu
<code>insert</code>	přidá údaj na zadaný index
<code>pop</code>	odstraní údaj na zadaném indexu
<code>remove</code>	odstraní zadaný údaj z listu
<code>reverse</code>	obrátí pořadí údajů v listu
<code>sort</code>	seřadí údaje v listu

```

1 # Metoda 'append' přidá údaj na konec listu
2 muj_seznam_1 = ["a", "b", "c"] # vytvořím novou proměnnou s hodnotou listu
3 muj_seznam_1.append(1) # přidám do listu 'int' 1
4 print(muj_seznam_1)

```

SPUSTIT ZNOVU

```
['a', 'b', 'c', 1]
```

```

1 # Metoda 'count' vrátí počet výskytu zadané hodnoty v listu
2 print([1, 1, 1].count(1)) # zjistíme, kolik jedniček list obsahuje

```

SPUSTIT ZNOVU

```

1 # Metoda 'sort' seřadí údaje v listu
2 muj_seznam_2 = ["b", "c", "a", "d"] # vytvořím novou proměnnou s hodnotou listu
3 muj_seznam_2.sort() # seřadím hodnoty pomocí metody 'sort'
4 print(muj_seznam_2)

```

[SPUSTIT ZNOVU](#)

['a', 'b', 'c', 'd']

## Metody tuplů

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / METODY / METODY TUPLŮ

### Metoda

[count](#)

### Použití

vrací počet výskytu zadané hodnoty v tuplu

### index

hledá v tuplu zadanou hodnotu a vrací index, na kterém najde hodnotu

```

1 # Metoda 'count' vrací počet výskytu zadané hodnoty v tuplu
2 print((1, 2, 1).count(1)) # zjistíme, kolik jedniček tuple obsahuje

```

[SPUSTIT ZNOVU](#)

```
1 # Metoda 'index' vrátí index, na kterém najde hledanou hodnotu
2 print("a", "b", "c").index("c"))
```

SPUSTIT ZNOVU

2

## Kvíz

PYTHON #1: ÚVOD DO PROGRAMOVÁNÍ / PODMÍNKY / METODY / KVÍZ

1/4

Jaký je správný zápis metody stringů `upper`?

- A. `str.upper('hodnota')`
- B. `'hodnota'.upper()`,
- C. `upper('hodnota')`
- D. `upper(str, 'hodnota')`

## Úkol 5: Vytvoření listu, přidávání prvků

**SPUSTIT ÚKOL**

Vytvoř list uchazečů a zaměstnanců a přidej do něj první prvky!

⌚ 15 min. ●●● těžší

## Úkol 6: Operace na listu 1

**SPUSTIT ÚKOL**

Procič si operace na listu - odstranění, opakování, spojování a indexing.

⌚ 15 min. ●●●● pokročilé

## Úkol 7: Operace na listu 2

**SPUSTIT ÚKOL**

Procič si operace na listu - slicing, striding, určování indexu a počtu prvků.

⌚ 20 min. ●●●● pokročilé

## Úkol 8: Palindrom

**SPUSTIT ÚKOL**

Víš, co jsou to palindromy? Napiš program, který taková slova rozpozná!

⌚ 10 min. ●●● střední

## Úkol 9: První písmeno

**SPUSTIT ÚKOL**

Vytvoř program, který bude ověřovat tvou znalost dnů v týdnu.

⌚ 15 min. ●●● střední

## Úkol 10: BMI kalkulačka

**SPUSTIT ÚKOL**

Změř si své BMI pomocí vlastního programu!

⌚ 20 min. ●●● těžší

## Úkol 11: Zjištění délky stringu

**SPUSTIT ÚKOL**

Program z této úlohy bude přijímat vstup od uživatele a odpovídat mu, kolik znaků obsahuje.

⌚ 15 min. ●●● střední

## Úkol 12: Kámen, nůžky, papír

**SPUSTIT ÚKOL**

Naprogramujeme si primitivní hru, kde budeme soutěžit s počítačem v souboji kámen, nůžky, papír.

⌚ 20 min. ●●● těžší

DALŠÍ LEKCE

>\_ Terminál

