Úvod do jazyka Python a počítačového programování

Modul 2

2.1 Literály - data sama o sobě

Nyní, když jste se seznámili s některými výkonnými funkcemi funkce print(), je čas seznámit se s některými novými problémy a jedním důležitým novým pojmem - literálem.

Literál je údaj, jehož hodnoty jsou určeny samotným literálem.

Protože se jedná o obtížně pochopitelný pojem, může vám pomoci dobrý příklad.

```
123
```

Uhodnete, jakou hodnotu představuje? Samozřejmě, že ano - je to sto dvacet tři.

Ale co tohle:

```
c
```

Představuje nějakou hodnotu? Možná. Může to být například symbol rychlosti světla. Může to být také integrační konstanta. Nebo dokonce délka přepony ve smyslu Pythagorovy věty. Možností je mnoho.

Bez dalších znalostí nelze vybrat tu správnou.

A to je vodítko: `123` je literál a `c` není.

Literály používáte k zakódování dat a k jejich vložení do kódu. Nyní si ukážeme některé konvence, které musíte při používání jazyka Python dodržovat.

Začneme jednoduchým experimentem - podívejte se na následující úryvek v editoru:

```
print("2")
print(2)
```

První řádek vypadá povědomě. Druhý se zdá být chybný kvůli viditelné absenci uvozovek.Zkuste jej spustit. Pokud vše proběhne v pořádku, uvidíte nyní dva stejné řádky.

```
2
2
```

Co se stalo? Co to znamená?

Prostřednictvím tohoto příkladu se setkáte se dvěma různými typy literálů:

- řetězec, který již znáte,
- a celým číslem, což je něco zcela nového

Funkce `print()` je prezentuje naprosto stejným způsobem - tento příklad je zřejmý, protože stejná je i jejich lidsky čitelná reprezentace. Vnitřně, v paměti počítače, jsou tyto dvě hodnoty uloženy zcela odlišným způsobem - řetězec existuje jen jako řetězec - řada písmen.

Číslo je převedeno na strojovou reprezentaci (sadu bitů). Funkce `print()` je schopna zobrazit obě ve formě čitelné pro člověka. Nyní se budeme nějaký čas věnovat číselným literálům a jejich vnitřnímu životu.

2.2 Integers (celá čísla)

Možná už něco málo víte o tom, jak počítače provádějí výpočty s čísly. Možná jste slyšeli o dvojkové soustavě a víte, že je to soustava, kterou počítače používají k ukládání čísel a že tyto počítače s nimi mohou provádět libovolné operace.

Nebudeme zde zkoumat složitosti pozičních číselných soustav, ale řekneme si, že čísla, se kterými pracují moderní počítače, jsou dvojího typu:

- celá čísla, tj. ta, která jsou zbavena zlomkové části
- čísla s pohyblivou řádovou čárkou (nebo jednoduše floaty), která obsahují (nebo mohou obsahovat) zlomkovou část

Tato definice není zcela přesná, ale pro tuto chvíli je zcela dostačující. Rozlišení je velmi důležité a hranice mezi těmito dvěma typy čísel je velmi přísná. Oba tyto druhy čísel se výrazně liší způsobem uložení v paměti počítače a rozsahem přípustných hodnot.

Vlastnost číselné hodnoty, která určuje její druh, rozsah a použití, se nazývá typ.

Pokud zakódujete literál a umístíte jej do kódu jazyka Python, tvar literálu určuje, jakou reprezentaci (typ) bude jazyk Python používat pro jeho uložení v paměti.

Prozatím ponechme stranou čísla s pohyblivou řádovou čárkou (brzy se k nim vrátíme) a věnujme se otázce, jak Python rozpoznává celá čísla.

Postup je téměř stejný, jako byste je psali tužkou na papír - je to prostě řetězec číslic, které tvoří číslo. Je tu však jedna výhrada - dovnitř čísla nesmíte vměstnat žádné znaky, které nejsou číslicemi.

Vezměme si například číslo jedenáct milionů sto jedenáct tisíc sto jedenáct. Kdybyste teď vzali do ruky tužku, napsali byste číslo takto: `11 111 111`, nebo takto: `11.111.111`, nebo dokonce takto: `11 111 111`.

Je jasné, že toto ustanovení usnadňuje čtení, zejména když se číslo skládá z mnoha číslic. Python však takovéto věci neakceptuje. Je to zakázáno. Co však Python povoluje, je použití podtržítek v číselných literálech.*.

Proto můžete toto číslo zapsat buď takto: `11111111`, nebo takto: `11_111_111`.

Poznámka: *Python 3.6 zavedl podtržítka v číselných literálech, což umožňuje umisťovat jednotlivá podtržítka mezi číslice a za specifikátory základu pro lepší čitelnost. Tato funkce není ve starších verzích jazyka Python k dispozici.

A jak v jazyce Python kódujeme záporná čísla? Jako obvykle - přidáním mínusu. Můžete napsat: `-11111111` nebo `-11_111_111`.

Kladným číslům nemusí předcházet znaménko plus, ale je to přípustné, pokud si to přejete. Následující řádky popisují stejné číslo: `+11111111` a `11111111`.

Osmičková a šestnáctková čísla

V jazyce Python existují dvě další konvence, které svět matematiky nezná. První nám umožňuje používat čísla v osmičkové reprezentaci.

Pokud celému číslu předchází předpona `0O` nebo `0o` (zero-o), bude s ním zacházeno jako s osmičkovou hodnotou. To znamená, že číslo musí obsahovat pouze číslice z rozsahu [0..7].

`0o123` je osmičkové číslo s (desetinnou) hodnotou `83`.

Funkce `print()` provede konverzi automaticky. Vyzkoušejte tento postup:

```
print(0o123)
```

Druhá konvence nám umožňuje používat hexadecimální čísla. Takovým číslům by měla předcházet předpona `0x` nebo `0X` (zero-x).

`0x123` je hexadecimální číslo s (desítkovou) hodnotou rovnou 291. S těmito hodnotami si poradí i funkce `print()`. Vyzkoušejte si následující postup:

```
print(0x123)
```

2.3 Floats (čísla s pohyblivou řádovou čárkou)

Nyní je čas promluvit si o dalším typu, který je určen k reprezentaci a ukládání čísel, která (jak by řekl matematik) mají neprázdný desetinný zlomek.

Jsou to čísla, která mají (nebo mohou mít) za desetinnou čárkou zlomkovou část, a i když je taková definice velmi chudá, pro to, o čem chceme diskutovat, je jistě dostačující.

Kdykoli použijeme termín jako dva a půl nebo minus nula celá čtyři, máme na mysli čísla, která počítač považuje za čísla s pohyblivou řádovou čárkou:

```
2.5
-0.4
```

Poznámka: dva a půl vypadá při zápisu v programu normálně, pokud však váš rodný jazyk preferuje používání čárky místo tečky v čísle, měli byste zajistit, aby vaše číslo čárky vůbec neobsahovalo.

Python to nebude akceptovat nebo (ve velmi vzácných, ale možných případech) může špatně pochopit váš záměr, protože čárka sama o sobě má v Pythonu svůj vlastní vyhrazený význam.

Pokud chcete použít právě hodnotu dva a půl, měli byste ji zapsat tak, jak je uvedeno výše. Ještě jednou si všimněte: mezi čísly 2 a 5 je tečka, nikoli čárka.

Jak si asi dokážete představit, hodnotu nula celá čtyři by bylo možné v jazyce Python zapsat takto:

0.4
Nezapomeňte však na toto jednoduché pravidlo - nulu můžete vynechat, pokud je jedinou číslicí před nebo za desetinnou čárkou.

V podstatě můžete hodnotu 0,4 zapsat jako:

```
.4
```

Například: hodnotu 4,0 lze zapsat jako:

```
4.
```

Tím se nezmění ani její typ, ani její hodnota.

Inty vs. floaty

Desetinná tečka je pro rozpoznání čísel s pohyblivou řádovou čárkou v jazyce Python nezbytná.

Podívejte se na tato dvě čísla:

```
4
4.0
```

Možná si myslíte, že jsou úplně stejná, ale Python je vnímá úplně jinak.

`4` je celé číslo, zatímco `4.0` je číslo s pohyblivou řádovou čárkou.

Tečka je to, co dělá float číslem.

Na druhou stranu, nejen tečky tvoří float. Můžete také použít písmeno `e`.

Když chcete použít nějaká čísla, která jsou velmi velká nebo velmi malá, můžete použít vědecký zápis.

Vezměme si například rychlost světla vyjádřenou v metrech za sekundu. Přímo zapsaná by vypadala takto: `300000000.`

Abyste nemuseli vypisovat tolik nul, používají učebnice fyziky zkrácený tvar, se kterým jste se již pravděpodobně setkali: 3×10^8.

To znamená: třikrát deset na osmou mocninu.

V jazyce Python se stejného efektu dosáhne trochu jiným způsobem - podívejte se jakým:

```
3E8
```

Písmeno E (můžete použít i malé písmeno e - pochází ze slova exponent) je výstižným zápisem věty krát deset na mocninu.

Pozn:

- *exponent* (hodnota za písmenem E) musí být celé číslo;
- *základ* (hodnota před písmenem E) může být buď celé číslo, nebo float.

Kódování floatů

Podívejme se, jak se tato konvence používá pro zápis čísel, která jsou velmi malá (ve smyslu jejich absolutní hodnoty, která se blíží nule).

Fyzikální konstanta, která se nazývá Planckova konstanta (a označuje se jako h), má podle učebnic hodnotu: 6.62607×10^{-34}.

Pokud byste ji chtěli použít v programu, měli byste ji zapsat takto:

```
6.62607E-34
```

Poznámka
skutečnost, že jste si vybrali jednu z možných forem zápisu hodnoty float, neznamená, že ji Python bude prezentovat stejným způsobem.

Python může někdy zvolit jiný zápis než vy.

Řekněme například, že jste se rozhodli použít následující float literál:

```
0.00000000000000000001
```

Když tento literál spustíte v Pythonu:

```
print(0.00000000000000000001)
```

toto je výsledek:

```
1e-22
```

Výstup: Python vždy volí úspornější formu prezentace čísla, což byste měli brát v úvahu při vytváření literálů.

2.4 Řetězce

Řetězce se používají, když potřebujete zpracovat text (například jména všeho druhu, adresy, romány atd.), nikoli čísla.

Něco málo o nich již víte, např. že řetězce potřebují uvozovky stejně jako floaty potřebují body.

Toto je velmi typický řetězec: `"Jsem řetězec."`

Má to však jeden háček. Ten háček spočívá v tom, jak zakódovat uvozovky uvnitř řetězce, který je již uvozovkami ohraničen.

Předpokládejme, že chceme vypsat velmi jednoduchou zprávu, která říká:

```
Mám rád "Monty Python"
```
Jak to uděláme, aniž bychom vygenerovali chybu? Existují dvě možná řešení.

První vychází z nám již známého konceptu znaku escape, který, jak si jistě pamatujete, hraje roli zpětného lomítka. Zpětné lomítko může uvozovky také escapovat. Citace, které předchází zpětné lomítko, mění svůj význam - není to oddělovač, ale pouze uvozovka. To bude fungovat tak, jak bylo zamýšleno:

```
print("Mám rád \"Monty Python\"")
```

Poznámka: uvnitř řetězce jsou dvě escapované uvozovky - vidíte je obě?

Druhé řešení může být trochu překvapivé. Python může místo uvozovek použít apostrof. Kterýkoli z těchto znaků může ohraničovat řetězce, ale musíte být důslední.

Pokud řetězec otevřete uvozovkou, musíte jej uvozovkou i uzavřít.

Pokud řetězec začínáte apostrofem, musíte jej ukončit apostrofem.

Tento příklad bude fungovat také:

```
print('Mám rád "Monty Python"').
```

Poznámka: zde nemusíte provádět žádné escapování.

Kódování řetězců

Další otázka nyní zní: jak vložíte apostrof do řetězce umístěného mezi apostrofy?

Odpověď byste již měli znát, přesněji řečeno dvě možné odpovědi.

Zkuste vypsat řetězec obsahující následující zprávu:

```
I'm Monty Python.
```

Víte, jak to udělat?

Zkontrolujte (ukázka řešení č. 1)

```
print('I\'m Monty Python.')
```

Zkontrolujte (ukázkové řešení č. 2)

```
print("I'm Monty Python.")
```

Jak vidíte, zpětné lomítko je velmi mocný nástroj - dokáže uniknout nejen uvozovkám, ale také apostrofům.

Už jsme si to ukázali, ale chceme tento jev zdůraznit ještě jednou: řetězec může být prázdný - nemusí obsahovat vůbec žádné znaky.

Prázdný řetězec stále zůstává řetězcem:

```
''
""
```

2.5 Logické hodnoty

Na závěr k literálům jazyka Python přidáme ještě dva další.

Nejsou tak zřejmé jako všechny předchozí, protože se používají k reprezentaci velmi abstraktní hodnoty - pravdivosti.

Pokaždé, když se Pythonu zeptáte, zda je jedno číslo větší než druhé, výsledkem této otázky je vytvoření nějakého konkrétního údaje - logické hodnoty.

Název pochází od George Boolea (1815-1864), autora základního díla *Zákony myšlení*, které obsahuje definici Booleovy algebry - části algebry, která využívá pouze dvě různé hodnoty: Pravda a nepravda, označované jako 1 a 0.

Programátor napíše program a ten mu klade otázky. Python program provede a poskytne odpovědi. Program musí být schopen reagovat podle obdržených odpovědí.

Počítače naštěstí znají pouze dva druhy odpovědí:

Ano, to je pravda;
Ne, toto je nepravda.

Nikdy nedostanete odpověď jako např: Nevím nebo Pravděpodobně ano, ale nevím to jistě.

Krajta je tedy binární plaz.

Tyto dvě logické hodnoty mají v jazyce Python striktní označení:

```
True
False
```

Nemůžete na nich nic měnit - musíte tyto symboly brát tak, jak jsou, včetně rozlišování velkých a malých písmen.

Úkol: Jaký bude výstup následujícího úryvku kódu?

```
print(True > False)
print(True < False)
```

Spusťte kód v editoru a zkontrolujte to. Dokážete vysvětlit výsledek?

```
True
False
```

LAB Python literály - řetězce

Scénář
Napište jednořádkový kus kódu s použitím funkce `print()` a znaků nový řádek a escape tak, aby odpovídal očekávanému výsledku vypsanému na třech řádcích.

Očekávaný výstup:

```
"I'm"
```

```
""learning""
"""Python"""
```

SHRNUTÍ SEKCE

1. Literály jsou zápisy pro reprezentaci některých pevných hodnot v kódu. Python má různé typy literálů - například literál může být číslo (číselný literál, např. `123`) nebo řetězec (řetězcový literál, např. `"Jsem literál."`).

2. Dvojková soustava je číselná soustava, která používá jako základ číslo 2. Binární číslo se tedy skládá pouze z nul a jedniček, např. `1010` je 10 v desítkové soustavě. Podobně osmičková a šestnáctková číselná soustava používají jako základ 8, resp. 16. Šestnáctková soustava používá desítková čísla a šest dalších písmen.

3. Celá čísla (nebo jednoduše **inty**) jsou jedním z číselných typů podporovaných jazykem Python. Jsou to čísla zapsaná bez zlomkové složky, např. `256` nebo `−1` (záporná celá čísla).

4. Čísla s pohyblivou řádovou čárkou (nebo jednoduše **floaty**) jsou dalším z číselných typů podporovaných jazykem Python. Jsou to čísla, která obsahují (nebo mohou obsahovat) zlomkovou složku, např. `1.27`.

5. Chcete-li zakódovat apostrof nebo uvozovku uvnitř řetězce, můžete buď použít znak escape, např. `'I\'m happy.'`, nebo řetězec otevřít a uzavřít pomocí opačné sady symbolů, než které chcete zakódovat, např. `"I'm happy."` pro zakódování apostrofu a `'He said "Python", not "typhoon"'` pro zakódování (dvojité) uvozovky.

6. Booleovské hodnoty jsou dva konstantní objekty `True` a `False`, které se používají k reprezentaci pravdivostních hodnot (v číselném kontextu je `1` je `True`, zatímco `0` je `False`.

Extra

V jazyce Python se používá ještě jeden speciální literál: literál `None`. Tento literál je objekt typu `NoneType` a používá se k reprezentaci nepřítomnosti hodnoty. Brzy si o něm povíme více.

KVÍZ

Otázka 1:
Jaké typy literálů představují následující dva příklady?

```
"Hello ", "007"
```

Otázka 2:
Jakými typy literálů jsou následující čtyři příklady?

```
"1.5", 2.0, 528, False
```

Otázka 3:

Jaká je desítková hodnota následujícího binárního čísla?

```
1011
```

Odkazy:

Cisco Programming Essentials in Python

Root.cz

ITNetwork.cz

Internet