Úvod do jazyka Python a počítačového programování

Modul 3

Logické hodnoty, podmíněné provádění, smyčky, seznamy a zpracování seznamů, logické a bitové operace

V tomto modulu se budete zabývat následujícími tématy:

- datový typ Boolean;
- relační operátory;
- rozhodování v jazyce Python (if, if-else, if-elif,else).
- jak opakovat provádění kódu pomocí cyklů (while, for)
- jak provádět logické a bitové operace v jazyce Python;
- seznamy v jazyce Python (konstrukce, indexování a řezání; manipulace s obsahem)
- jak třídit seznamy pomocí algoritmů bublinového třídění;
- vícerozměrné seznamy a jejich použití.

3.1.1.1 Rozhodování v jazyce Python

Otázky a odpovědi

Programátor napíše program a ten mu klade otázky.

Počítač program spustí 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 pravda není.

Nikdy nedostanete odpověď typu Nech mě přemýšlet...., nevím, nebo Pravděpodobně ano, ale nevím to jistě.

K pokládání otázek používá Python sadu velmi speciálních operátorů. Projdeme si je jeden po druhém a ukážeme si jejich účinky na několika jednoduchých příkladech.

Porovnání: operátor rovnosti

Otázka: Jsou si dvě hodnoty rovny?

K položení této otázky se používá operátor == (rovná se).

Nezapomeňte na tento důležitý rozdíl:

- = je operátor přiřazení, např. a = b přiřadí a hodnotu b;
- == je otázka, zda se tyto hodnoty rovnají; a == b porovnává a a b.

Je to binární operátor s levostrannou vazbou. Potřebuje dva argumenty a kontroluje, zda se rovnají.

Otázka č. 1: Jaký je výsledek následujícího porovnání?

2 == 2

Pravda - samozřejmě, 2 se rovná 2. Python odpoví True (pamatujte na tuto dvojici předdefinovaných literálů True a False - jsou to také klíčová slova Pythonu).

Otázka č. 2: Jaký je výsledek následujícího porovnání?

2 == 2.

Tato otázka není tak snadná jako první. Python naštěstí umí převést celočíselnou hodnotu na její reálný ekvivalent, a proto je odpověď True.

Otázka č. 3: Jaký je výsledek následujícího porovnání?

1 == 2

Tohle by mělo být snadné. Odpověď bude (nebo spíše vždy bude) False.

3.1.1.2 Rozhodování v jazyce Python

Operátor `==` (rovná se) porovnává hodnoty dvou operandů. Pokud se rovnají, je výsledkem porovnání hodnota `True`. Pokud se nerovnají, výsledkem porovnání je `False`.

Podívejte se na porovnání rovnosti níže - jaký je výsledek této operace?

```
var == 0
```

Všimněte si, že odpověď nemůžeme zjistit, pokud nevíme, jaká hodnota je aktuálně uložena v proměnné `var`.

Pokud byla proměnná během provádění programu mnohokrát změněna nebo je její počáteční hodnota zadána z konzoly, může odpověď na tuto otázku dát pouze Python a pouze za běhu.

Nyní si představte programátora, který trpí nespavostí a musí počítat černé a bílé ovce zvlášť, dokud je černých ovcí přesně dvakrát více než bílých.

Otázka bude znít následovně:

```
black_sheep == 2 * white_sheep
```

Vzhledem k nízké prioritě operátoru == se tato otázka považuje za rovnocennou:

```
black_sheep == (2 * white_sheep)
```

Pojďme si nyní procvičit porozumění operátoru == - dokážete odhadnout výstup níže uvedeného kódu?

```
var = 0  # Assigning 0 to var
print(var == 0)

var = 1  # Assigning 1 to var
print(var == 0)
```

Spusťte kód a zkontrolujte, zda jste měli pravdu.

Nerovnost: operátor nerovná se (!=)

Operátor != (nerovná se) také porovnává hodnoty dvou operandů. Zde je rozdíl: pokud se rovnají, výsledkem porovnání je False. Pokud se nerovnají, je výsledkem porovnání True.

Nyní se podívejte na porovnání nerovností níže - dokážete odhadnout výsledek této operace?

```
var = 0  # Assigning 0 to var
print(var != 0)

var = 1  # Assigning 1 to var
print(var != 0)
```

Spusťte kód a zkontrolujte, zda jste měli pravdu.

Operátory porovnávání: větší než

Můžete také položit srovnávací otázku pomocí operátoru > (větší než). Pokud chcete zjistit, zda je více černých ovcí než bílých, můžete ji zapsat takto:

```
black_sheep > white_sheep  # Greater than
```

`True` to potvrzuje, `False` to popírá.

Operátory porovnávání: větší než nebo rovno

Operátor větší než má ještě jednu speciální, nestriktní variantu, která se však značí jinak než v klasickém aritmetickém zápisu: >= (větší než nebo rovno).

Následná znaménka jsou dvě, nikoliv jedno.

Oba tyto operátory (striktní i nestriktní), stejně jako dva další, o nichž bude řeč v následující části, jsou **binární operátory s levostrannou vazbou** a jejich **priorita je větší než priorita**, kterou vykazují operátory `==` a `!=`.

Chceme-li zjistit, zda máme či nemáme nosit teplou čepici, položíme si následující otázku:

```
centigrade_outside >= 0.0  # Greater than or equal to
```

Operátory porovnávání: menší než nebo rovno

Jak jste již pravděpodobně uhodli, operátory použité v tomto případě jsou: operátor `<` (méně než) a jeho nepřísný sourozenec: `<=` (méně než nebo rovno).

Podívejte se na tento jednoduchý příklad:

```
current_velocity_mph < 85  # Less than
current_velocity_mph ≤ 85  # Less than or equal to
```

Zjistíme, zda hrozí pokuta od dálniční policie (první otázka je přísná, druhá ne).

Využití odpovědí

Co můžete udělat s odpovědí (tj. s výsledkem porovnávací operace), kterou získáte z počítače?

Existují přinejmenším dvě možnosti: zaprvé si ji můžete zapamatovat (uložit do proměnné) a využít ji později. Jak to uděláte? No, použijete libovolnou proměnnou, například takto:

```
answer = number_of_lions >= number_of_lionesses
```

Obsah proměnné vám prozradí odpověď na položenou otázku.

Druhá možnost je pohodlnější a mnohem častější: na základě získané odpovědi můžete rozhodnout o budoucnosti programu. K tomuto účelu potřebujete speciální instrukci, kterou si brzy probereme.

Nyní musíme aktualizovat naši tabulku priorit a zařadit do ní všechny nové operátory. Ta nyní vypadá následovně:

Priorita	Operátor	
1	`+`, `-`	unární
2	`**`	
3	`*`, `/`, `//`, `%`	
4	`+`, `-`	binární
5	`<`, `<=`, `>`, `>=`	
6	`==`, `!=`	

Pozn:

`**` - Exponentiation

`//` - floor division - zaokrouhlí výsledek dělení na nejbližší celé číslo směrem dolů

`%` - modulus (zbytek po celočíselném dělení)

LAB

Odhadovaný čas

5-10 minut

Úroveň obtížnosti

Velmi snadné

Cíle

seznámení se s funkcí input(); seznámení se s operátory porovnávání v jazyce Python.

Scénář

Pomocí některého z operátorů porovnávání v jazyce Python napište jednoduchý dvouřádkový program, který jako vstup přijme parametr n, což je celé číslo, a vypíše False, pokud je n menší než 100, a True, pokud je n větší nebo rovno 100.

Nevytvářejte žádné bloky if (o nich budeme hovořit velmi brzy). Otestujte svůj kód pomocí dat, která jsme vám poskytli.

Testovací data

Sample input: `55`

Expected output: `False`

Sample input: `99`

Expected output: `False`

Sample input: `100`

Expected output: `True`

Sample input: `101`

Expected output: `True`

Sample input: `-5`

Expected output: `False`

Sample input: `+123`

Expected output: `True`

3.1.1.5 Rozhodování v jazyce Python

Podmínky a podmíněné provádění

Už víte, jak klást otázky Pythonu, ale stále nevíte, jak rozumně využít odpovědi. Musíte mít mechanismus, který vám umožní něco udělat, pokud je podmínka splněna, a neudělat to, pokud splněna není.

Je to jako ve skutečném životě: určité věci děláte, nebo neděláte, když je splněna určitá podmínka, nebo ne, např. jdete na procházku, když je dobré počasí, nebo zůstanete doma, když je mokro a zima.

K takovému rozhodování nabízí Python speciální instrukci. Vzhledem k její povaze a použití se nazývá podmíněná instrukce (nebo podmíněný příkaz).

Existuje několik jejích variant. Začneme tou nejjednodušší, přičemž obtížnost budeme pomalu zvyšovat.

První podoba podmíněného příkazu, kterou vidíte níže, je zapsána velmi neformálně, ale obrazně:

```
if true_or_not:
    do_this_if_true
```

Tento podmíněný výrok se skládá z následujících nezbytně nutných prvků, a to pouze v tomto a tomto pořadí:

- klíčové slovo `if`;
- jedna nebo více bílých mezer;
- výraz (otázka nebo odpověď), jehož hodnota bude interpretována výhradně ve smyslu `True` (když je jeho hodnota nenulová) a `False` (když je rovna nule);
- dvojtečka následovaná novým řádkem;
- odsazená instrukce nebo sada instrukcí (alespoň jedna instrukce je bezpodmínečně nutná); odsazení lze dosáhnout dvěma způsoby - vložením určitého počtu mezer (doporučuje se použít čtyři mezery odsazení) nebo použitím znaku tabulátoru; pozn. překl: Pokud je v odsazené části více než jedna instrukce, mělo by být odsazení ve všech řádcích stejné; i když může vypadat stejně, pokud použijete tabulátory smíchané s mezerami, je důležité, aby všechna odsazení byla naprosto stejná - Python 3 nedovoluje pro odsazení míchat mezery a tabulátory.

Jak tato konstrukce funguje?

Pokud výraz `true_or_not` **představuje pravdu** (tj. jeho hodnota není rovna nule), provede se odsazený příkaz (příkazy);
pokud výraz `true_or_not` **nepředstavuje pravdu** (tj. jeho hodnota je rovna nule), odsazený příkaz (příkazy) se vynechá (ignoruje) a další prováděná instrukce bude ta, která následuje po původní úrovni odsazení.

V reálném životě často vyjadřujeme přání:

Pokud bude pěkné počasí, půjdeme na procházku.

pak půjdeme na oběd

if the weather is good, we'll go for a walk

then, we'll have lunch

Jak vidíte, oběd není podmíněný a nezávisí na počasí.

Pokud víme, jaké podmínky ovlivňují naše chování, a předpokládáme, že máme k dispozici bezparametrické funkce `go_for_a_walk()` a `have_lunch()`, můžeme napsat následující úryvek:

```
if the_weather_is_good:
    go_for_a_walk()
have_lunch()
```

3.1.1.6 Rozhodování v jazyce Python

Pokud jistý nespavý vývojář v Pythonu usne, když napočítá 120 oveček, a postup navozující spánek může být implementován jako speciální funkce s názvem `sleep_and_dream()`, má celý kód následující podobu:

```
if sheep_counter >= 120: # Evaluate a test expression
    sleep_and_dream() # Execute if test expression is True
```

Můžete to číst jako: pokud je `sheep_counter` větší nebo rovno 120, pak usni a sni (tj. proveď funkci `sleep_and_dream`).

Řekli jsme, že podmíněně prováděné příkazy musí být odsazeny. Vzniká tak velmi čitelná struktura, která jasně demonstruje všechny možné cesty provádění kódu.

Podívejte se na následující kód:

```
if sheep_counter >= 120:
    make_a_bed()
    take_a_shower()
    sleep_and_dream()
feed_the_sheepdogs()
```

Jak vidíte, stlaní postele, sprchování, usínání a snění **se provádí podmíněně** - když sheep_counter dosáhne požadované hranice.

Krmení ovčáků se však **provádí vždy** (tj. funkce `feed_the_sheepdogs()` není odsazena a nepatří do bloku `if`, což znamená, že se provádí vždy).

Nyní probereme další variantu podmíněného příkazu, která rovněž umožňuje provést další akci, když podmínka není splněna.

Podmíněné provádění: příkaz if-else

Začali jsme jednoduchou větou, která zněla: Pokud bude dobré počasí, půjdeme na procházku.

Všimněte si - není zde ani slovo o tom, co se stane, když bude špatné počasí. Víme pouze, že nepůjdeme ven, ale co bychom mohli dělat místo toho, není známo. Možná si budeme chtít naplánovat něco i pro případ špatného počasí.

Můžeme si například říct: Pokud bude dobré počasí, půjdeme na procházku, v opačném případě půjdeme do divadla.

Nyní víme, co budeme dělat, **pokud budou splněny podmínky**, a víme, co budeme dělat, **pokud se vše nevydaří podle našich představ**. Jinými slovy, máme "**plán B**".

Python nám umožňuje takové alternativní plány vyjádřit. K tomu slouží druhá, o něco složitější forma podmíněného příkazu, příkaz if-else:

```
if true_or_false_condition:
    perform_if_condition_true
else:
    perform_if_condition_false
```

Je zde nové slovo: `else` - jedná se o klíčové slovo.

Část kódu, která začíná slovem `else`, říká, co se má udělat, pokud není splněna podmínka zadaná pro `if` (všimněte si dvojtečky za tímto slovem).

Provedení if-else probíhá následovně:

- Pokud se podmínka vyhodnotí jako **`True`** (její hodnota není rovna nule), provede se příkaz `perform_if_condition_true` a podmíněný příkaz se ukončí;
- pokud se podmínka vyhodnotí jako **`False`** (její hodnota je rovna nule), provede se příkaz `perform_if_condition_false` a podmíněný příkaz se ukončí.

3.1.1.7 Rozhodování v jazyce Python

Příkaz if-else: další podmíněné provádění

Pomocí této formy podmíněného výroku můžeme naše plány popsat následovně:

```
if the_weather_is_good:
    go_for_a_walk()
else:
    go_to_a_theater()
```

```
have_lunch()
```

Pokud bude dobré počasí, půjdeme na procházku. Jinak půjdeme do divadla. Bez ohledu na to, zda je počasí dobré nebo špatné, půjdeme potom na oběd (po procházce nebo po návštěvě divadla).

Vše, co jsme si řekli o odsazení, funguje stejným způsobem i uvnitř větného členu `else`:

```
if the_weather_is_good:
    go_for_a_walk()
    have_fun()
else:
 go_to_a_theater()
 enjoy_the_movie()
have_lunch()
```

Vnořené příkazy if-else

Nyní probereme dva speciální případy podmíněného příkazu. Nejprve se zamysleme nad případem, kdy **instrukcí umístěnou za příkazem** `if` je další příkaz `if`.

Přečtěte si, co máme v plánu na tuto neděli. Pokud bude pěkné počasí, půjdeme na procházku. Pokud najdeme pěknou restauraci, dáme si tam oběd. V opačném případě si dáme sendvič. Pokud bude špatné počasí, půjdeme do divadla. Pokud nebudou lístky, půjdeme nakupovat do nejbližšího obchodního centra.

Napišme totéž v jazyce Python. Pečlivě zvažte kód, který zde uvádíme:

```
if the_weather_is_good:
 if nice_restaurant_is_found:
     have_lunch()
 else:
     eat_a_sandwich()
else:
 if tickets_are_available:
     go_to_the_theater()
 else:
     go_shopping()
```

```
Zde jsou dva důležité body:
```

Toto použití příkazu `if` se nazývá vnořování; nezapomeňte, že každé else se vztahuje k příkazu `if`, který leží na stejné úrovni odsazení; musíte to vědět, abyste určili, jak se příkazy `if` a `else` párují; zvažte, jak odsazení zlepšuje čitelnost a usnadňuje pochopení a sledování kódu.

Odkazy:

Cisco Programming Essentials in Python

Root.cz

ITNetwork.cz

Internet