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.

Příkaz elif

Druhý speciální případ zavádí další nové klíčové slovo jazyka Python: `elif`. Jak asi tušíte, jedná se o kratší formu `else if`.

Příkaz elif se používá k ověření více než jen jedné podmínky a k zastavení, když je nalezen první příkaz, který je pravdivý.

Náš další příklad se podobá vnořování, ale podobnost je velmi malá. Opět změníme naše plány a vyjádříme je takto: Pokud bude pěkné počasí, půjdeme na procházku, jinak pokud dostaneme lístky, půjdeme do divadla, jinak pokud budou v restauraci volné stoly, půjdeme na oběd; pokud všechno ostatní selže, zůstaneme doma a budeme hrát šachy.

Všimli jste si, kolikrát jsme použili slovo jinak? V této fázi hraje svou roli klíčové slovo elif.

Napišme stejný scénář pomocí jazyka Python:

```
if the_weather_is_good:
    go_for_a_walk()
elif tickets_are_available:
    go_to_the_theater()
elif table_is_available:
    go_for_lunch()
else:
    play_chess_at_home()
```

Způsob sestavování následných příkazů `if-elif-else` se někdy nazývá kaskáda.

Opět si všimněte, jak odsazení zlepšuje čitelnost kódu.

V tomto případě je třeba věnovat určitou pozornost navíc:

- nesmíte použít `else` bez předchozího `if`;
- `else` je vždy poslední větví kaskády, bez ohledu na to, zda jste použili `elif` nebo ne;
- `else` je nepovinnou součástí kaskády a lze ji vynechat;
- pokud je v kaskádě větev `else`, provede se pouze jedna ze všech větví;
- pokud větev `else` neexistuje, je možné, že se neprovede žádná z dostupných větví.

Může to znít trochu záhadně, ale doufejme, že několik jednoduchých příkladů pomůže osvětlit tuto problematiku.

1.7 Analýza vzorků kódu

Nyní vám ukážeme několik jednoduchých, ale kompletních programů. Nebudeme je podrobně vysvětlovat, protože komentáře (a názvy proměnných) uvnitř kódu považujeme za dostatečné vodítko.

Všechny programy řeší stejný problém - najdou největší z několika čísel a vypíší je.

Příklad 1:

Začneme nejjednodušším případem - jak určit větší ze dvou čísel:

```
# Read two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Choose the larger number
```

```
if number1 > number2:
    larger_number = number1
else:
    larger_number = number2

# Print the result
print("The larger number is:", larger_number)
```

Výše uvedený úryvek by měl být jasný - načte dvě celočíselné hodnoty, porovná je a zjistí, která z nich je větší.

Příklad 2:

Nyní vám ukážeme jednu zajímavou skutečnost. Python má zajímavou vlastnost - podívejte se na kód níže:

```
# Read two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# Choose the larger number
if number1 > number2: larger_number = number1
else: larger_number = number2

# Print the result
print("The larger number is:", larger_number)
```

Poznámka:
pokud některá z větví `if-elif-else` obsahuje pouze jednu instrukci, můžete ji zakódovat v obsáhlejší podobě (nemusíte dělat odsazený řádek za klíčovým slovem, ale stačí pokračovat v řádku za dvojtečkou).

Tento styl však může být zavádějící a v našich budoucích programech ho používat nebudeme, ale rozhodně stojí za to ho znát, pokud chcete číst a rozumět cizím programům.

Jiné rozdíly v kódu nejsou.

Příklad 3:

Je čas zkomplikovat kód - najdeme největší ze tří čísel. Zvětší se tím kód? Trochu.

Předpokládáme, že první hodnota je největší. Poté tuto hypotézu ověříme u zbývajících dvou hodnot.

Podívejte se na kód níže:

```
# Read three numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))

# We temporarily assume that the first number
```

```
# is the largest one.
# We will verify this soon.
largest_number = number1

# We check if the second number is larger than the current
largest_number
# and update the largest_number if needed.
if number2 > largest_number:
    largest_number = number2

# We check if the third number is larger than the current
largest_number
# and update the largest_number if needed.
if number3 > largest_number:
    largest_number = number3

# Print the result
print("The largest number is:", largest_number)
```

Tato metoda je podstatně jednodušší než hledání největšího čísla najednou porovnáváním všech možných dvojic čísel (tj. prvního s druhým, druhého s třetím, třetího s prvním). Zkuste si kód přestavět sami.

1.8 Pseudokód a úvod do smyček

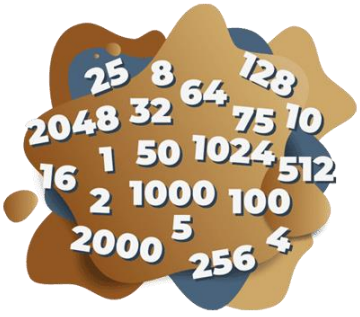
Nyní byste měli být schopni napsat program, který najde největší ze čtyř, pěti, šesti nebo dokonce deseti čísel.

Schéma již znáte, takže rozšíření velikosti problému nebude nijak zvlášť složité.

Co se ale stane, když vás požádáme, abyste napsali program, který najde největší ze dvou set čísel? Dokážete si představit takový kód?

Budete potřebovat dvě stě proměnných. Pokud vám dvě stě proměnných nestačí, zkuste si představit hledání největšího z milionu čísel.

Představte si kód, který obsahuje 199 podmíněných příkazů a dvě stě volání funkce input(). Naštěstí se tím nemusíte zabývat. Existuje jednodušší přístup.



Požadavky na syntaxi jazyka Python budeme prozatím ignorovat a pokusíme se analyzovat problém bez přemýšlení o skutečném programování. Jinými slovy, pokusíme se napsat algoritmus, a až s ním budeme spokojeni, budeme ho implementovat.

V tomto případě použijeme jakýsi zápis, který sice není skutečným programovacím jazykem (nelze jej zkompilovat ani spustit), ale je formalizovaný, stručný a čitelný. Říká se mu pseudokód.

Podívejme se na náš pseudokód níže:

```
největší_číslo = -999999999
number = int(input())
if number == -1:
    print(největší_číslo)
    exit()
if number > largest_number:
    největší_číslo = číslo
# Přejděte na řádek 02
```

Co se v něm děje?

Nejprve můžeme program zjednodušit, pokud hned na začátku kódu přiřadíme proměnné největší_číslo hodnotu, která bude menší než kterékoli ze zadaných čísel. K tomuto účelu použijeme -999999999.

Za druhé předpokládáme, že náš algoritmus nebude předem vědět, kolik čísel bude do programu dodáno. Předpokládáme, že uživatel zadá tolik čísel, kolik bude chtít - algoritmus bude dobře pracovat se stovkou i s tisícovkou čísel. Jak to uděláme?

Uzavřeme s uživatelem dohodu: když zadá hodnotu -1, bude to znamení, že už nejsou žádná další data a program má ukončit svou práci.

V opačném případě, pokud se zadaná hodnota nebude rovnat -1, program přečte další číslo atd.

Tento trik vychází z předpokladu, že jakoukoli část kódu lze provést více než jednou - přesněji řečeno tolikrát, kolikrát je potřeba.

Provedení určité části kódu více než jednou se nazývá smyčka. Význam tohoto termínu je vám pravděpodobně zřejmý.

Řádky 02 až 08 tvoří smyčku. Budeme jimi procházet tolikrát, kolikrát bude potřeba, abychom zkontrolovali všechny zadané hodnoty.

Dokážete podobnou strukturu použít v programu napsaném v jazyce Python? Ano, můžete.

Další informace

Python často obsahuje spoustu vestavěných funkcí, které udělají práci za vás. Chcete-li například zjistit největší číslo ze všech, můžete použít vestavěnou funkci Pythonu s názvem `max()`. Můžete ji použít s více argumenty. Analyzujte níže uvedený kód:

```
# Read three numbers.
```

```
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))

# Check which one of the numbers is the greatest
# and pass it to the largest_number variable.

largest_number = max(number1, number2, number3)

# Print the result.
print("The largest number is:", largest_number)
```

Stejným způsobem můžete použít funkci `min()` pro vrácení nejnižšího čísla. Výše uvedený kód můžete přestavět a experimentovat s ním v prostředí Sandbox.

O těchto (a mnoha dalších) funkcích budeme brzy hovořit. Prozatím se zaměříme na podmíněné provádění a cykly, abyste získali větší jistotu v programování a naučili se dovednostem, které vám umožní plně pochopit a aplikovat tyto dva koncepty v kódu. Prozatím tedy nepůjdeme žádnou zkratkou.

LAB Operátory porovnávání a podmíněné provádění

Spathiphyllum, známější jako lilie pokojová nebo bílá plachta, je jednou z nejoblíbenějších pokojových rostlin, která filtruje škodlivé toxiny ze vzduchu. Mezi toxiny, které neutralizuje, patří benzen, formaldehyd a čpavek.

Představte si, že váš počítačový program tyto rostliny miluje. Kdykoli obdrží vstup v podobě slova Spathiphyllum, nedobrovolně vykřikne do konzole následující řetězec: "Spathiphyllum je nejlepší rostlina na světě!"

Napište program, který využívá koncepci podmíněného provádění, přijímá jako vstup řetězec a:

vypisuje větu "Ano - Spathiphyllum je nejlepší rostlina na světě".

na obrazovku, pokud je na vstupu řetězec "Spathiphyllum" (velká písmena).

vypíše na obrazovku větu "Ne, chci velké Spathiphyllum!", pokud je zadaný řetězec "spathiphyllum" (malá písmena).

vypíše "Spathiphyllum! Ne [vstup]!" v opačném případě. Poznámka: [input] je řetězec přijatý jako vstup.

Otestujte svůj kód pomocí dat, která jsme vám poskytli. A pořiďte si také Spathiphyllum!

Testovací data:

Vstupní vzorek:	**Očekávaný výstup:**
spathiphyllum	Ne, chci velké Spathiphyllum!
pelargonium	Spathiphyllum! Ne pelargonie!
Spathiphyllum	Ano - Spathiphyllum je nejlepší rostlina na světě!

Začátek kódu např.:

```
name = input("Enter flower name: ")
```

LAB Základy příkazu if-else

Scénář

Byla jednou jedna země - země mléka a strdí, obývaná šťastnými a prosperujícími lidmi. Lidé samozřejmě platili daně - jejich štěstí mělo své hranice. Nejdůležitější daň, nazývaná daň z příjmu fyzických osob (zkráceně PIT), se musela platit jednou ročně a vyměřovala se podle následujícího pravidla:

- Pokud příjem občana nebyl vyšší než 85 528 tolarů, daň se rovnala 18 % příjmu minus 556 tolarů a 2 centy (tomu se říkalo daňová úleva).
- pokud byl příjem vyšší než tato částka, daň se rovnala 14 839 tolarům a 2 centům plus 32 % z přebytku nad 85 528 tolarů.

Vaším úkolem je napsat daňovou kalkulačku.

Měla by přijímat jednu hodnotu s pohyblivou řádovou čárkou: příjem.

Dále by měla vypsat vypočtenou daň zaokrouhlenou na celé tolary. Existuje funkce `round()`, která zaokrouhlování provede za vás - najdete ji v kostře kódu v editoru.

Poznámka: tato šťastná země nikdy nevrátila svým občanům žádné peníze. Pokud by vypočtená daň byla menší než nula, znamenalo by to pouze to, že daň nebyla vůbec žádná (daň se rovnala nule). Berte to při výpočtech v úvahu.

Podívejte se na kód v editoru - načte pouze jednu vstupní hodnotu a vypíše výsledek, takže je třeba jej doplnit o chytré výpočty.

Vyzkoušejte svůj kód na základě následujících údajů:

Vstupní vzorek:	**Očekávaný výstup:**
10000	Daň je: 1244,0 tolarů
100000	Daň je: 19470,0 tolarů
1000	Daň je: 0,0 tolaru
-100	Daň je: 0,0 tolaru

Vzorový kód:

```
income = float(input("Enter the annual income: "))
if income < 85528:
      tax = income * 0.18 - 556.02
# Write the rest of your code here.
tax = round(tax, 0)
print("The tax is:", tax, "thalers")
```

Odkazy:

Cisco Programming Essentials in Python

Root.cz

ITNetwork.cz

Internet