Úvod do jazyka Python a počítačového programování

Modul 3 sekce 2 – Smyčky (cykly) v jazyce Python

Ve druhé části se seznámíte s cykly v jazyce Python, konkrétně s cykly while a for. Dozvíte se, jak vytvářet nekonečné smyčky (a jak se vyhnout jejich pádu), jak ukončovat smyčky a přeskakovat jednotlivé iterace smyček. Připraveni?

2.1 Smyčky (cykly) kódu pomocí `while`

Souhlasíte s níže uvedeným tvrzením?

```
While (dokud) je potřeba něco udělat
    dělej to
```

Všimněte si, že tento záznam také prohlašuje, že pokud není co dělat, nestane se vůbec nic.

Obecně lze v jazyce Python smyčku reprezentovat následujícím způsobem:

```
while
    instrukce
```

Pokud si všimnete některých podobností s instrukcí `if`, je to zcela v pořádku. Syntaktický rozdíl je skutečně jen jeden: místo slova `if` používáte slovo `while`.

Sémantický rozdíl je důležitější: když je splněna podmínka, `if` provede své příkazy pouze jednou; `while` opakuje provádění tak dlouho, dokud je podmínka vyhodnocena jako `True`.

Poznámka: i zde platí všechna pravidla týkající se odsazování. To si brzy ukážeme.

Podívejte se na následující algoritmus:

```
while conditional_expression:
    instruction_one
    instruction_two
    instruction_three
    :
    :
    instruction_n
```

Nyní je důležité mít na paměti, že:

pokud chcete uvnitř jedné smyčky while provést více než jeden příkaz, musíte (stejně jako u příkazu if) všechny instrukce odsadit stejným způsobem;

instrukce nebo sada instrukcí prováděná uvnitř cyklu while se nazývá tělo cyklu;

pokud je podmínka False (rovna nule) již při prvním testování, tělo se neprovede ani jednou (všimněte si analogie, že nemusíte nic dělat, když není co dělat);

tělo by mělo mít možnost měnit hodnotu podmínky, protože pokud je podmínka na začátku True, mohlo by tělo běžet nepřetržitě až do nekonečna - všimněte si, že provedení nějaké věci obvykle snižuje počet věcí, které je třeba provést).

2.2 Nekonečná smyčka (cyklus)

Nekonečná smyčka, nazývaná také nekonečná smyčka, je posloupnost instrukcí v programu, která se opakuje donekonečna (nekonečná smyčka).

Zde je příklad smyčky, která není schopna dokončit své provádění:

```
while True:
    print("Uvízl jsem uvnitř smyčky.")
```

Tato smyčka bude donekonečna vypisovat na obrazovku `"Uvízl jsem uvnitř smyčky."`.

Poznámka
Pokud se chcete co nejlépe naučit, jak se chová nekonečná smyčka, spusťte IDLE, vytvořte Nový soubor, zkopírujte a vložte výše uvedený kód, uložte soubor a spusťte program. Uvidíte nekonečnou sekvenci řetězců "Uvízl jsem uvnitř smyčky." vypisovaných do okna konzoly Pythonu. Chcete-li program ukončit, stačí stisknout klávesu Ctrl-C (nebo Ctrl-Break na některých počítačích). To způsobí výjimku KeyboardInterrupt a umožní vašemu programu dostat se ze smyčky. O tom si povíme později v tomto kurzu.

Vraťme se k náčrtu algoritmu, který jsme vám nedávno ukázali. Ukážeme si, jak tuto nově naučenou smyčku použít k nalezení největšího čísla z velkého souboru zadaných dat.
Pozorně si program analyzujte. Podívejte se, kde začíná smyčka (řádek 8). Najděte tělo smyčky a zjistěte, jak je tělo ukončeno:

```
# Store the current largest number here.
largest_number = -999999999

# Input the first value.
number = int(input("Enter a number or type -1 to stop: "))

# If the number is not equal to -1, continue.
while number != -1:
    # Is number larger than largest_number?
    if number > largest_number:
        # Yes, update largest_number.
        largest_number = number
    # Input the next number.
    number = int(input("Enter a number or type -1 to stop: "))

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

Podívejte se, jak tento kód implementuje algoritmus, který jsme vám ukázali dříve.

2.3 Smyčka (cyklus) `while`: další příklady

Podívejme se na další příklad s využitím cyklu while. Sledujte komentáře, abyste zjistili myšlenku a řešení.

```
# A program that reads a sequence of numbers
# and counts how many numbers are even and how many are odd.
# The program terminates when zero is entered.

odd_numbers = 0
even_numbers = 0

# Read the first number.
number = int(input("Enter a number or type 0 to stop: "))

# 0 terminates execution.
while number != 0:
    # Check if the number is odd.
    if number % 2 == 1:
        # Increase the odd_numbers counter.
        odd_numbers += 1
    else:
        # Increase the even_numbers counter.
        even_numbers += 1
    # Read the next number.
    number = int(input("Enter a number or type 0 to stop: "))

# Print results.
print("Odd numbers count:", odd_numbers)
print("Even numbers count:", even_numbers)
```

Některé výrazy lze zjednodušit, aniž by se změnilo chování programu.

Zkuste si připomenout, jak Python interpretuje pravdivost podmínky, a všimněte si, že tyto dvě formy jsou ekvivalentní:

`while number != 0:` a `while number:`

Podmínku, která kontroluje, zda je číslo liché, lze zakódovat také v těchto ekvivalentních formách:

`if number % 2 == 1:` a `if number % 2:`

Použití proměnné čítače k ukončení smyčky

Podívejte se na úryvek kódu níže:

```
counter = 5
while counter != 0:
    print("Uvnitř smyčky.", counter)
    counter -= 1
print("Mimo smyčku.", counter)
```

Tento kód má za úkol vypsat řetězec "Uvnitř smyčky." a hodnotu uloženou v proměnné `counter` během dané smyčky přesně pětkrát. Jakmile není podmínka splněna (proměnná `counter` dosáhla hodnoty 0), smyčka se ukončí a vypíše se zpráva "Mimo smyčku." a hodnota uložená v `counter`.

Jednu věc však lze zapsat kompaktněji - podmínku cyklu `while`.

Vidíte ten rozdíl?

```
counter = 5
while counter:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

Je kompaktnější než dříve? Trochu. Je čitelnější? To je sporné.

PAMATUJTE SI

Neciťte se povinni kódovat své programy tak, aby byly vždy nejkratší a nejkompaktnější. Čitelnost může být důležitějším faktorem. Udržujte svůj kód připravený pro nového programátora.

LAB Uhodněte tajné číslo

Mladší kouzelník si vybral tajné číslo. Schoval ho do proměnné s názvem secret_number. Chce, aby si každý, kdo spustí jeho program, zahrál hru Hádej tajné číslo a uhodl, jaké číslo mu vybral. Ti, kteří číslo neuhodnou, uvíznou v nekonečné smyčce navždy! Bohužel neví, jak kód dokončit.

Vaším úkolem je pomoci kouzelníkovi doplnit kód v editoru tak, aby kód:

- požádá uživatele o zadání celého čísla;
- bude používat smyčku while;
- zkontroluje, zda číslo zadané uživatelem je stejné jako číslo vybrané kouzelníkem. Pokud se číslo zvolené uživatelem liší od kouzelníkova tajného čísla, měl by uživatel vidět zprávu "Ha ha! Uvízl jsi v mé smyčce!" a bude vyzván k opětovnému zadání čísla. Pokud se číslo zadané uživatelem shoduje s číslem vybraným kouzelníkem, mělo by se číslo vytisknout na obrazovku a kouzelník by měl říci následující slova: "Dobrá práce, mudlo! Nyní jsi volný."

Kouzelník s vámi počítá! Nezklamte ho.

EXTRA INFO

Mimochodem, podívejte se na funkci `print()`. Způsob, který jsme zde použili, se nazývá víceřádkový tisk. Pomocí trojitých uvozovek můžete vytisknout řetězce na více řádků, aby byl text lépe čitelný, nebo vytvořit speciální textový design. Experimentujte s ním.

```
secret_number = 777

print(
"""
+================================+
| Welcome to my game, muggle!    |
| Enter an integer number        |
| and guess what number I've     |
| picked for you.                |
| So, what is the secret number? |
+================================+

""")
```

Odkazy:

Cisco Programming Essentials in Python

Root.cz

ITNetwork.cz

Internet