Úvod do jazyka Python a počítačového programování

Modul 3 sekce 2 - Smyčky 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 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

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 `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? |
+================================+

""")
```

2.4 Cykly kódu pomocí `for`

Další druh smyčky, který je v jazyce Python k dispozici, vychází z pozorování, že někdy je důležitější počítat "otočky" smyčky než kontrolovat podmínky.

Představte si, že tělo smyčky je třeba provést přesně stokrát. Pokud byste k tomu chtěli použít cyklus while, může vypadat takto:

```
i = 0
while i < 100:
    # do_something()
    i += 1
```

Bylo by hezké, kdyby to nudné počítání někdo udělal za vás. Je to možné?

Samozřejmě, že ano - pro tyto typy úloh existuje speciální smyčka, která se jmenuje `for`.

Ve skutečnosti je smyčka `for` určena k řešení složitějších úloh - dokáže "procházet" velké kolekce dat položku po položce. Brzy si ukážeme, jak na to, ale nyní si představíme jednodušší variantu jeho použití.

Podívejte se na úryvek kódu:

```
for i in range(100):
    # do_something()
    pass
```

Jsou zde některé nové prvky. Dovolte nám, abychom vám o nich pověděli:

Klíčové slovo `for` otevírá smyčku `for`; všimněte si, že za ním není žádná podmínka; na podmínky nemusíte myslet, protože jsou kontrolovány interně, bez jakéhokoli zásahu;

jakákoli proměnná za klíčovým slovem `for` je řídicí proměnnou smyčky; počítá otáčky smyčky, a to automaticky;

klíčové slovo `in` zavádí syntaktický prvek popisující rozsah možných hodnot, které se přiřazují řídicí proměnné;

funkce `range()` (jedná se o velmi speciální funkci) je zodpovědná za generování všech požadovaných hodnot řídicí proměnné; v našem příkladu funkce vytvoří (můžeme dokonce říci, že bude smyčku krmit) hodnotami z následující množiny: Poznámka: v tomto případě funkce `range()` začíná svou práci od 0 a končí ji o jeden krok (jedno celé číslo) před hodnotou svého argumentu;

všimněte si klíčového slova `pass` uvnitř těla cyklu - nedělá vůbec nic; je to prázdná instrukce - dáváme ji sem proto, že syntaxe cyklu for vyžaduje alespoň jednu instrukci uvnitř těla (mimochodem - `if`, `elif`, `else` a `while` vyjadřují totéž)

Naše další příklady budou v počtu opakování smyčky o něco skromnější.

Podívejte se na úryvek níže. Dokážete předpovědět jeho výstup?

```
for i in range(10):
    print("The value of i is currently", i)
```

Spusťte kód a zkontrolujte, zda jste měli pravdu.

Poznámka:

smyčka byla provedena desetkrát (je to argument funkce `range()`).

Hodnota poslední řídicí proměnné je 9 (nikoliv 10, protože začíná od 0, nikoliv od 1)

Volání funkce `range()` může být vybaveno dvěma argumenty, nikoliv pouze jedním:

```
for i in range(2, 8):
    print("The value of i is currently", i)
```

V tomto případě první argument určuje počáteční (první) hodnotu řídicí proměnné.
Poslední argument udává první hodnotu, která nebude řídicí proměnné přiřazena.

Poznámka: funkce `range()` přijímá jako argumenty pouze celá čísla a generuje posloupnosti celých čísel.

Dokážete odhadnout výstup programu? Spusťte jej a zkontrolujte, zda jste se také trefili.
První zobrazená hodnota je 2 (převzatá z prvního argumentu funkce `range()`.
Poslední hodnota je 7 (přestože druhý argument funkce `range()` je 8).

2.5 Více o cyklu `for` a funkci `range()` se třemi argumenty

Funkce `range()` může také přijímat tři argumenty - podívejte se na kód v editoru.

```
for i in range(2, 8, 3):
    print("The value of i is currently", i)
```

Třetím argumentem je inkrement - hodnota přidaná ke kontrole proměnné při každém otočení smyčky (jak možná tušíte, výchozí hodnota inkrementu je 1).

Můžete nám říci, kolik řádků se objeví v konzoli a jaké hodnoty budou obsahovat?
Spusťte program a zjistěte, zda máte pravdu.
V okně konzoly byste měli vidět následující řádky:

The value of i is currently 2
The value of i is currently 5

Víte proč? První argument předaný funkci `range()` nám říká, jaké je počáteční číslo posloupnosti (proto 2 ve výstupu). Druhý argument říká funkci, kde má posloupnost zastavit (funkce generuje čísla až do čísla uvedeného druhým argumentem, ale nezahrnuje ho). A konečně třetí argument udává krok, což vlastně znamená rozdíl mezi jednotlivými čísly v posloupnosti čísel generovaných funkcí.

2 (počáteční číslo) → 5 (přírůstek 2 o 3 se rovná 5 - číslo je v rozsahu od 2 do 8) → 8 (přírůstek 5 o 3 se rovná 8 - číslo není v rozsahu od 2 do 8, protože parametr stop není zahrnut v posloupnosti čísel generovaných funkcí).

Poznámka: pokud je množina vygenerovaná funkcí `range()` prázdná, smyčka své tělo vůbec neprovede.

Stejně jako zde - nebude žádný výstup:

```
for i in range(1, 1):
    print("Hodnota i je aktuálně", i)
```

Poznámka:
množina vytvořená funkcí `range()` musí být seřazena vzestupně. Neexistuje žádný způsob, jak donutit funkci `range()` vytvořit množinu v jiném tvaru, pokud funkce `range()` přijímá přesně dva argumenty. To znamená, že druhý argument funkce `range()` musí být větší než první.

Ani zde tedy nedojde k žádnému výstupu:

```
for i in range(2, 1):
    print("The value of i is currently", i)
```

Podívejme se na krátký program, jehož úkolem je zapsat některou z prvních mocnin dvou:

```
power = 1
for expo in range(16):
    print("2 to the power of", expo, "is", power)
    power *= 2
```

Proměnná expo slouží jako řídicí proměnná smyčky a udává aktuální hodnotu exponentu. Samotná exponence je nahrazena násobením dvěma. Protože 20 je rovno 1, pak 2 × 1 je rovno 21, 2 × 21 je rovno 22 atd. Jaký je největší exponent, pro který náš program ještě vypíše výsledek?

Spusťte kód a zkontrolujte, zda výstup odpovídá vašemu očekávání.

LAB Základy smyčky `for` - počítání mississippily

Scénář

Víte, co je Mississippi? No, je to název jednoho ze států a jedné z řek ve Spojených státech. Délka řeky Mississippi je přibližně 2340 mil, což z ní činí druhou nejdelší řeku ve Spojených státech (nejdelší je řeka Missouri). Je tak dlouhá, že jediná kapka vody potřebuje 90 dní, aby překonala celou její délku!

Slovo Mississippi se používá také k trochu jinému účelu: k počítání mississippily.

Pokud tento výraz neznáte, vysvětlíme vám, co znamená: používá se k počítání vteřin.

Její podstata spočívá v tom, že když při hlasitém počítání vteřin přidáte k číslu slovo mississippi, bude znít blíže k času hodin, a proto "jedna mississippi, dvě mississippi, tři mississippi" bude trvat přibližně skutečné tři vteřiny času! Často ji používají děti při hře na schovávanou, aby se ujistily, že hledač počítá poctivě.

Váš úkol je zde velmi jednoduchý: napište program, který pomocí smyčky `for` "počítá mississippily" do pěti. Po napočítání do pěti by měl program na obrazovku vypsat závěrečnou zprávu "Připraven nebo ne, už jdu!".

Použijte kostru, kterou jsme poskytli v editoru.

EXTRA INFO
Všimněte si, že kód v editoru obsahuje dva prvky, které vám v tuto chvíli nemusí být zcela jasné: příkaz `import time` a metodu `sleep()`. Brzy si o nich něco povíme.

Prozatím bychom chtěli, abyste věděli jen to, že jsme importovali modul time a použili metodu `sleep()` k pozastavení provádění každé následující funkce `print()` uvnitř cyklu for na jednu sekundu, takže zpráva vypisovaná na konzoli připomíná skutečné počítání. Nebojte se - brzy se o modulech a metodách dozvíte více.

Očekávaný výstup:

1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi

Kostra programu:

```
Import time
# Napište smyčku for, která počítá do pěti.
    # Tělo cyklu - vypište číslo iterace cyklu a slovo "Mississippi".
    # Tělo smyčky - použijte: time.sleep(1)

# Napište funkci print s konečnou zprávou.
```

2.6 Příkazy `break` a `continue`

Doposud jsme tělo smyčky považovali za nedělitelnou a neoddělitelnou posloupnost instrukcí, které se kompletně provádějí při každém otočení smyčky. Jako vývojář však můžete být postaveni před následující možnosti:

Zdá se, že je zbytečné pokračovat ve smyčce jako celku; měli byste se zdržet dalšího provádění těla smyčky a pokračovat dále;
zdá se, že je třeba zahájit další obrat smyčky, aniž byste dokončili provádění aktuálního obratu.
Python poskytuje dvě speciální instrukce pro realizaci obou těchto úloh. Pro přesnost řekněme, že jejich existence v jazyce není nutná - zkušený programátor je schopen nakódovat jakýkoli algoritmus i bez těchto instrukcí. Takovým doplňkům, které nezlepšují vyjadřovací schopnosti jazyka, ale pouze zjednodušují práci vývojáře, se někdy říká syntaktický bonbónek nebo syntaktický cukr.

Tyto dva pokyny jsou:

`break` - okamžitě ukončí smyčku a bezpodmínečně ukončí její činnost; program začne vykonávat nejbližší instrukci za tělem smyčky;
`continue` - chová se, jako by program náhle dosáhl konce těla; okamžitě se spustí další cyklus a testuje se výraz podmínky.
Obě tato slova jsou klíčová.

Nyní si ukážeme dva jednoduché příklady, které ilustrují, jak tyto dvě instrukce fungují. Podívejte se na kód v editoru. Spusťte program a analyzujte výstup. Upravte kód a experimentujte.

```
# break - example

print("The break instruction:")
for i in range(1, 6):
    if i == 3:
        break
    print("Inside the loop.", i)
print("Outside the loop.")

# continue - example

print("\nThe continue instruction:")
for i in range(1, 6):
    if i == 3:
        continue
    print("Inside the loop.", i)
print("Outside the loop.")
```

Příkazy `break` a `continue`: další příklady

Vraťme se k našemu programu, který rozpozná největší ze zadaných čísel. Převedeme jej dvakrát, a to pomocí příkazů `break` a `continue`.

Analyzujte kód a posuďte, zda a jak byste některý z nich použili.

Varianta `break` přichází:

```
largest_number = -99999999
counter = 0

while True:
    number = int(input("Enter a number or type -1 to end the program: "))
    if number == -1:
        break
    counter += 1
    if number > largest_number:
        largest_number = number

if counter != 0:
    print("The largest number is", largest_number)
else:
    print("You haven't entered any number.")
```

A nyní varianta `continue`:

```
largest_number = -99999999
counter = 0

number = int(input("Enter a number or type -1 to end program: "))

while number != -1:
    if number == -1:
        continue
```

```
    counter += 1

    if number > largest_number:
        largest_number = number
    number = int(input("Enter a number or type -1 to end the program: "))

if counter:
    print("The largest number is", largest_number)
else:
  print("You haven't entered any number.")
```

Podívejte se pozorně, uživatel zadá první číslo ještě předtím, než program vstoupí do smyčky `while`. Další číslo se zadává, když už je program ve smyčce.

Opět - spusťte program, vyzkoušejte ho a experimentujte s ním.

LAB Příkaz `break` - Uvíznutí ve smyčce

Scénář

Příkaz `break` slouží k ukončení smyčky.

Navrhněte program, který používá smyčku `while` a neustále žádá uživatele o zadání slova, pokud uživatel nezadá jako tajné výstupní slovo "chupacabra"; v takovém případě by se měla na obrazovku vypsat zpráva "Úspěšně jste opustili smyčku." a smyčka by se měla ukončit.

Nevypisujte žádné ze slov zadaných uživatelem. Použijte koncept podmíněného provádění a příkaz `break`.

LAB Příkaz `continue` - Ošklivý požírač samohlásek

Scénář

Příkaz `continue` slouží k přeskočení aktuálního bloku a k přechodu na další iteraci, aniž by se provedly příkazy uvnitř smyčky.

Lze jej použít jak s cykly `while`, tak s cykly `for`.

Váš úkol zde je velmi zvláštní: musíte navrhnout samohláskový požírač! Napište program, který používá:

smyčku `for`;

koncept podmíněného provádění (`if-elif-else`)

příkaz `continue`.

Váš program musí:

požádat uživatele o zadání slova;

použít `user_word` = `user_word.upper()` k převodu uživatelem zadaného slova na velká písmena; o metodách řetězců a metodě `upper()` budeme hovořit velmi brzy - nebojte se;

použijte podmíněné provádění a příkaz `continue`, abyste ze zadaného slova "sežrali" následující hlásky A, E, I, O, U;

vypsat nesežraná písmena na obrazovku, každé na samostatném řádku.

Otestujte svůj program s daty, která jsme vám poskytli.

Testová data:

Příklad vstupu:	Očekávaný výstup:
Gregory	G R G R Y
abstemious	B S T M S
IOUEA	

Kostra programu:

```
# Prompt the user to enter a word
# and assign it to the user_word variable.

for letter in user_word:
    # Complete the body of the for loop.
```

LAB Příkaz `continue` - Hezký požírač samohlásek

Váš úkol je zde ještě speciálnější než dříve: musíte přepracovat (ošklivý) samohláskový požírač z předchozí laboratoře a vytvořit lepší, vylepšený (hezký) samohláskový požírač! Napište program, který používá:

smyčku `for`;

koncept podmíněného provádění (`if-elif-else`).

příkaz `continue`.

Váš program musí:

požádat uživatele o zadání slova;

použít `user_word = user_word.upper()` k převodu uživatelem zadaného slova na velká písmena; o metodách řetězců a metodě `upper()` budeme hovořit velmi brzy - nebojte se;

použijte podmíněné provádění a příkaz continue, abyste ze zadaného slova "sežrali" následující hlásky A, E, I, O, U;

přiřaďte nesežraná písmena do proměnné `word_without_vowels` a vypište proměnnou na obrazovku.

Podívejte se na kód v editoru. Vytvořili jsme proměnnou `word_without_vowels` a přiřadili jí prázdný řetězec. Pomocí operace spojování požádejte Python, aby během dalších otočení smyčky spojil vybraná písmena do delšího řetězce, a přiřaďte jej do proměnné `word_without_vowels`.

Otestujte svůj program s daty, která jsme vám poskytli.

Testová data:

Příklad vstupu:	Očekávaný výstup:
Gregory	GRGRY
abstemious	BSTMS
IOUEA	

Kostra programu:

```
word_without_vowels = ""

# Prompt the user to enter a word
# and assign it to the user_word variable.


for letter in user_word:
    # Complete the body of the loop.

# Print the word assigned to word_without_vowels.
```

2.7 Smyčka `while` a větev `else`

Oba cykly, `while` i `for`, mají jednu zajímavou (a zřídka používanou) vlastnost.

Ukážeme vám, jak funguje - zkuste sami posoudit, zda je použitelná a zda se bez ní obejdete, nebo ne. Jinými slovy, zkuste se sami přesvědčit, zda je tato funkce cenná a užitečná, nebo je to jen syntaktický cukr.

Podívejte se na úryvek v editoru. Na konci je něco zvláštního - klíčové slovo `else`.

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

Jak jste možná tušili, větev else mohou mít i cykly, stejně jako `if`y.

Větev `else` smyčky se provede vždy jednou, bez ohledu na to, zda smyčka vstoupila do svého těla, nebo ne. Dokážete odhadnout, jaký bude výstup? Spusťte program a zkontrolujte, zda jste měli pravdu.

Trochu upravte úryvek tak, aby smyčka neměla šanci vykonat své tělo ani jednou:

```
i = 5
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)
```

Stav `while` je na začátku `False` - vidíte to?
Spusťte a vyzkoušejte program a zkontrolujte, zda byla provedena větev `else`, nebo ne.

2.8 Smyčka `for` a větev `else`

Smyčky `for` se chovají trochu jinak - podívejte se na úryvek v editoru a spusťte jej**.**

```
for i in range(5):
    print(i)
else:
    print("else:", i)
```

Výstup může být trochu překvapivý.
Proměnná i si zachovává svou poslední hodnotu.
Trochu upravte kód a proveďte ještě jeden experiment.

```
i = 111
for i in range(2, 1):
    print(i)
else:
    print("else:", i)
```

Dokážete odhadnout výstup?

Tělo smyčky se zde vůbec neprovede. Všimněte si: proměnnou `i` jsme přiřadili před smyčku.

Spusťte program a zkontrolujte jeho výstup.

Když se tělo smyčky nevykoná, řídicí proměnná si zachová hodnotu, kterou měla před smyčkou.

Poznámka: pokud řídicí proměnná neexistuje před spuštěním smyčky, nebude existovat ani v okamžiku, kdy provádění smyčky dospěje do větve `else`.

Co si myslíte o této variantě `else`?

Brzy si povíme něco o dalších druzích proměnných. Naše současné proměnné mohou uchovávat vždy jen jednu hodnotu, ale existují proměnné, které toho umí mnohem více - mohou uchovávat libovolný počet hodnot. Nejdříve si ale uděláme pár pokusů.

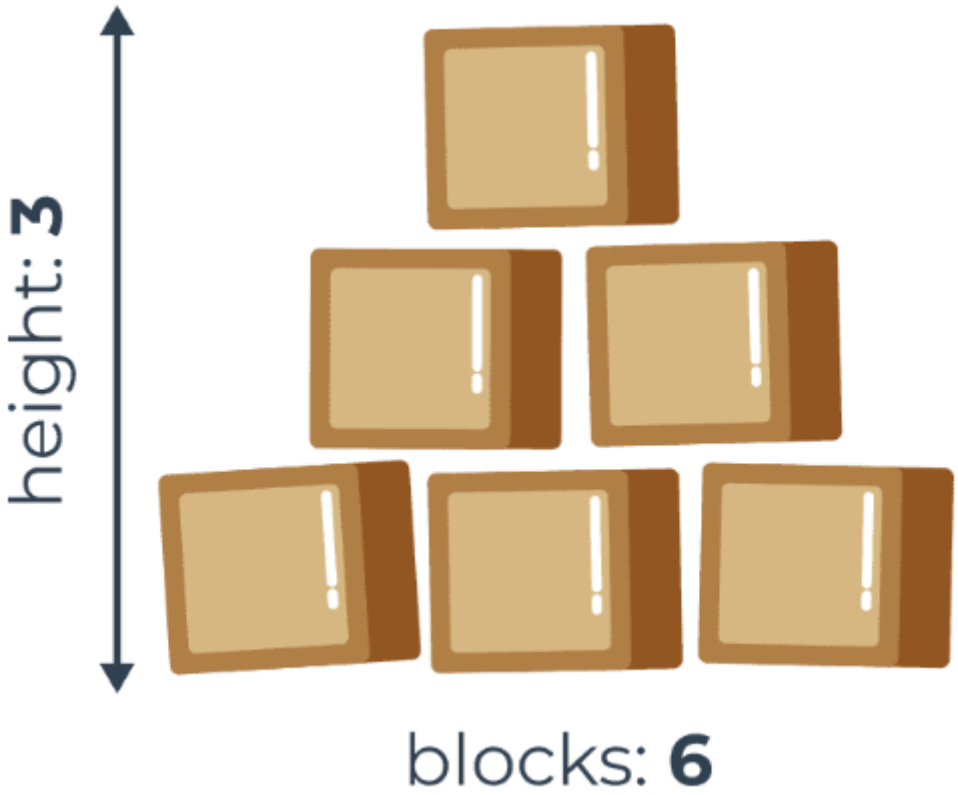
LAB Základy smyčky `while`

Scénář

Poslechněte si tento příběh: Chlapec a jeho otec, počítačový programátor, si hrají s dřevěnými kostkami. Staví pyramidu.

Jejich pyramida je trochu zvláštní, protože je to vlastně stěna ve tvaru pyramidy - je plochá. Pyramida se skládá podle jednoho jednoduchého principu: každá spodní vrstva obsahuje o jeden blok více než vrstva nad ní.

Obrázek znázorňuje pravidlo, které stavitelé použili:



Vaším úkolem je napsat program, který přečte počet kvádrů, které mají stavitelé, a vypíše výšku pyramidy, kterou lze z těchto kvádrů postavit.

Poznámka: výška se měří počtem zcela dokončených vrstev - pokud stavitelé nemají dostatečný počet bloků a nemohou dokončit další vrstvu, okamžitě svou práci ukončí.

Otestujte svůj kód s použitím údajů, které jsme vám poskytli.

Testová data:

Příklad vstupu:	Očekávaný výstup:
6	The height of the pyramid: 3
20	The height of the pyramid: 3
1000	The height of the pyramid: 44
2	The height of the pyramid: 1

```
blocks = int(input("Enter the number of blocks: "))

#
# Write your code here.
#

print("The height of the pyramid:", height)
```

LAB Collatzova hypotéza

Scénář

V roce 1937 formuloval německý matematik Lothar Collatz zajímavou hypotézu (dodnes neprokázanou), kterou lze popsat následujícím způsobem:

1. Vezměte libovolné nezáporné a nenulové celé číslo a pojmenujte ho c0;
2. pokud je sudé, vyhodnoťte nové c0 jako c0 ÷ 2;
3. v opačném případě, je-li liché, vyhodnoťte nové c0 jako 3 × c0 + 1;
4. pokud c0 ≠ 1, vraťte se k bodu 2.

Hypotéza říká, že bez ohledu na počáteční hodnotu c0 bude vždy 1.

Samozřejmě je nesmírně složitá úloha použít počítač k tomu, aby se hypotéza dokázala pro libovolné přirozené číslo (může to dokonce vyžadovat umělou inteligenci), ale k ověření některých jednotlivých čísel můžete použít Python. Možná dokonce najdete takové, které by hypotézu vyvrátilo.

Napište program, který načte jedno přirozené číslo a provede výše uvedené kroky, dokud c0 zůstane různé od 1. Chceme také, abyste spočítali kroky potřebné k dosažení cíle. Váš kód by měl také vypsat všechny mezihodnoty c0.

Tip: nejdůležitější částí úlohy je, jak převést Collatzovu myšlenku do cyklu while - to je klíč k úspěchu.

Vyzkoušejte svůj kód s použitím dat, která jsme vám poskytli.

Testová data:

Příklad vstupu:	Očekávaný výstup:

```
15
```

```
46
46
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1
steps = 17
```

```
16
```

```
8
4
2
1
steps = 4
```

```
1023
```

```
3070
1535
4606
2303
6910
3455
10366
5183
15550
7775
23326
11663
34990
17495
52486
26243
78730
39365
118096
59048
29524
14762
7381
22144
11072
5536
2768
1384
692
346
173
173
260
130
65
196
98
49
148
74
37
37
56
28
14
7
22
11
34
17
52
26
13
40
20
10
5
16
8
4
2
steps = 62
```

SHRNUTÍ SEKCE

1. V jazyce Python existují dva typy cyklů: while a for:
    - Smyčka `while` provádí příkaz nebo sadu příkazů tak dlouho, dokud je zadaná logická podmínka pravdivá, např.:

```
# Example 1
while True:
    print("Stuck in an infinite loop.")

# Example 2
counter = 5
while counter > 2:
    print(counter)
    counter -= 1
```

    - Smyčka `for` provádí množinu příkazů mnohokrát; používá se k iteraci nad posloupností (např. seznamem, slovníkem, tuplem nebo množinou - o nich se brzy dozvíte) nebo jinými iterovatelnými objekty (např. řetězci). Cyklus `for` můžete použít k iteraci nad posloupností čísel pomocí vestavěné funkce `range`. Podívejte se na příklady níže:

```
# Example 1
word = "Python"
for letter in word:
    print(letter, end="*")


# Example 2
for i in range(1, 10):
    if i % 2 == 0:
        print(i)
```

1. Pomocí příkazů break a continue můžete měnit průběh smyčky:
    - Příkazem break ukončíte smyčku, např.:

```
text = "OpenEDG Python Institute"
for letter in text:
    if letter == "P":
        break
    print(letter, end="")
```

    - Pro přeskočení aktuální iterace a pokračování v další iteraci se používá příkaz continue, např.:

```
text = "pyxpyxpyx
for letter in text:
    if letter == "x":
        continue
    print(letter, end="")
```

3. Smyčky while a for mohou mít v Pythonu také klauzuli `else`. Klauzule `else` se provede po ukončení provádění cyklu, pokud nebyl ukončen klauzulí `break`, např:

```
n = 0

while n != 3:
    print(n)
    n += 1
else:
    print(n, "else")

print()

for i in range(0, 3):
    print(i)
else:
    print(i, "else")
```

4. Funkce `range()` generuje posloupnost čísel. Přijímá celá čísla a vrací objekty rozsahu. Syntaxe funkce `range()` vypadá takto: `range(start, stop, step)`, kde:

   - `start` je nepovinný parametr určující počáteční číslo posloupnosti (ve výchozím nastavení 0)
   - `stop` je nepovinný parametr určující konec generované posloupnosti (není zahrnut),
   - a `step` je nepovinný parametr určující rozdíl mezi čísly v posloupnosti (ve výchozím nastavení 1).

Příklad kódu:

```
for i in range(3):
    print(i, end=" ")  # Outputs: 0 1 2

for i in range(6, 1, -2):
    print(i, end=" ")  # Outputs: 6, 4, 2
```

KVÍZ ODDÍLU

Otázka 1: Vytvořte cyklus `for`, který počítá od 0 do 10 a vypisuje na obrazovku lichá čísla. Použijte níže uvedenou kostru:

```
for i in range(1, 11):
    # Line of code.
        # Line of code.
```

Otázka 2: Vytvořte cyklus `while`, který počítá od 0 do 10 a vypisuje na obrazovku lichá čísla. Použijte níže uvedenou kostru:

```
x = 1
while x < 11:
    # Line of code.
        # Line of code.
    # Line of code.
```

Otázka 3: Vytvořte program se smyčkou `for` a příkazem `break`. Program by měl iterovat přes znaky v e-mailové adrese, ukončit cyklus, když dojde k symbolu @, a vypsat část před @ na jeden řádek. Použijte níže uvedenou kostru:

```
for ch in "john.smith@pythoninstitute.org":
    if ch == "@":
        # Line of code.
    # Line of code.
```

Otázka 4: Vytvořte program se smyčkou `for` a příkazem `continue`. Program by měl iterovat řetězec číslic, nahradit každou 0 za x a vypsat upravený řetězec na obrazovku. Použijte níže uvedenou kostru:

```
for digit in "0165031806510":
    if digit == "0":
        # Line of code.
        # Line of code.
    # Line of code.
```

Otázka 5: Jaký je výstup následujícího kódu?

```
n = 3

while n > 0:
    print(n + 1)
    n -= 1
else:
    print(n)
```

Otázka 6: Jaký je výstup následujícího kódu?

```
n = range(4)

for num in n:
    print(num - 1)
else:
    print(num)
```

Otázka 7: Jaký je výstup následujícího kódu?

```
for i in range(0, 6, 3):
    print(i)
```

Odkazy:

Cisco Programming Essentials in Python

Root.cz

ITNetwork.cz

Internet