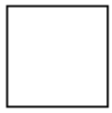
Rekurze

Jedním z důležitých principů pro návrh procedur je tzv. **rekurze**. Nejlépe uvidíme tento princip na příkladech dvou velmi jednoduchých procedur (hvězdička označuje násobení).

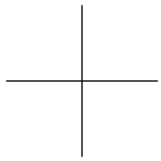
```
? to mnohouhelnik1 :a :uhel
> forward :a
> right :uhel
> mnohouhelnik1 :a :uhel
> end
Nadefinoval jsi mnohouhelnik1

? to mnohouhelnik2 :a :uhel
> forward :a
> right :uhel
> forward :a
> right (2 * :uhel)
> mnohouhelnik2 :a :uhel
> end
Nadefinoval jsi mnohouhelnik2
```

```
? mnohouhelnik1 50 90
```



```
? mnohouhelnik2 50 90
```

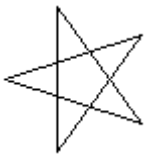


Rekurze vlastně označuje procedury, které předtím, než ukončí svou činnost, volají **samy sebe.** To ovšem znamená, že jejich činnost není nikdy ukončena! Nemusí tomu tak být, ale prozatím se s tím můžeme smířit a běh procedur ukončíme podle potřeby tlačítkem **STOP.**


```
? mnohouhelnik1 20 90
Běh zastavený na 2. řádku procedury mnohouhelnik1
```

A nyní zpět k uvedeným procedurám. Obě kreslí jakési obecné n-úhelníky, což lze vyzkoušet volbou různých parametrů pro úhel. Pro **mnohouhelnik1** vyzkoušejme dvojice hodnot [72 144], [1 60] a [135 108], pro **mnohouhelnik2** [30 144] a [45 125].

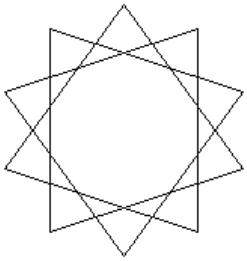
```
? mnohouhelnik1 72 144
```



```
? mnohouhelnik1 1 60
```

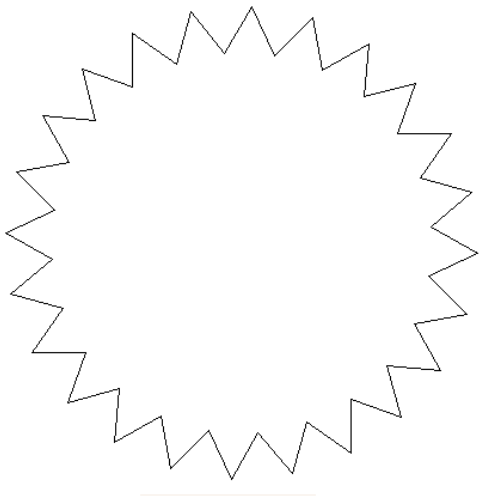


```
? mnohouhelnik1 135 108
```



```
? mnohouhelnik2 30 144
```

```
? mnohouhelnik2 45 125
```



Zkusme i jiné hodnoty pro úhel v předchozích procedurách, i desetinná čísla, např. 12,5 apod. A nakonec ještě jedno vylepšení. Představme si, že při rekurzivním volání procedury **mnohouhelnik1** zvětšíme velikost. To způsobí, že se bude místo obecného n-úhelníka kreslit jakási zvětšující se „spirála". Vyzkoušejme:

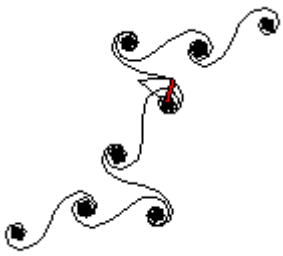
```
? to spirala :a :uhel :zvetseni
> forward :a
> right :uhel
> spirala (:a + :zvetseni) :uhel :zvetseni
> end
Nadefinoval jsi spirala
```

Zvolme velikost :a 10, zvětšení 1 a dosazujte úhel 90, 95, 120, 117 apod.. Situaci bychom mohli zopakovat pro zvětšování úhlu:

```
? to ispirala :a :uhel :zvetseni
> forward :a
> right :uhel
> ispirala :a (:uhel + :zvetseni) :zvetseni
> end
Nadefinoval jsi ispirala
```

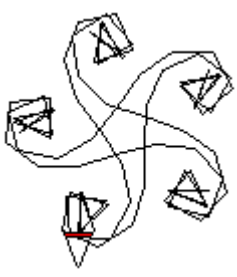
Účinek této procedury je překvapivý. Zvolme parametry pro úhel i zvětšení např. takto: [0 7], [40 30], [2 20]. To už vypadá téměř jako čarování, ale jsou za tím pouze matematické zákonitosti.

```
? ispirala 5 0 7
```



```
? ispirala 10 40 30
```

```
? ispirala 20 2 20
```



Želva jako zvíře

Prozatím jsme vždy popsali pohyb želvy zcela přesně, což znamená, že výsledkem procedury byl vždy stejný obrázek. Tak se ale zvíře nechová, protože v přírodě hraje jistou roli náhoda.

Procedura s návratovou hodnotou

V programovacích jazycích bývá začleněn tzv. generátor pseudonáhodných čísel, který umí na požádání vygenerovat náhodné číslo v nějakém rozsahu. V jazyku LOGO je k dispozici příkaz **random**, jehož výsledkem je náhodné nezáporné celé číslo mezi nulou a parametrem. Např. příkaz **random 10** vrátí náhodné (pokaždé jiné) číslo od nuly do devíti. Všimněme si ovšem, že pokud tento příkaz použijeme, LOGO vypíše zprávu o tom, že neví, co má dělat s výsledkem. To je nová situace, protože dosud nikdy nebyla výsledkem příkazu hodnota (prozatím totiž všechny procedury pracovaly tak, že jejich výsledkem byl pouze pohyb želvy). Hodnotu lze použít například pro pohyb dopředu

```
? forward (random 10)
```

nebo ji pouze zobrazit jako výsledek. K zobrazení výsledku slouží procedura **print,** tedy např.

```
? print (random 10)
6
? print (random 10)
3
? print (random 10)
1
```

Procedury, jejichž výsledkem je hodnota, se někdy označují jako **funkce.**

Příklad

Napišme proceduru, která vrátí jako výsledek číslo z nějakého intervalu. Ke vrácení hodnoty jako výsledku procedury slouží příkaz **output**, který zároveň ukončí proceduru. To je nejlépe vidět na zápisu procedury:

```
? to interval :od :do
> output (:od + random (:do - :od + 1))
> end
Nadefinoval jsi interval


? print (interval 10 20)
19
? print (interval 10 20)
10
? print (interval 10 20)
13
```

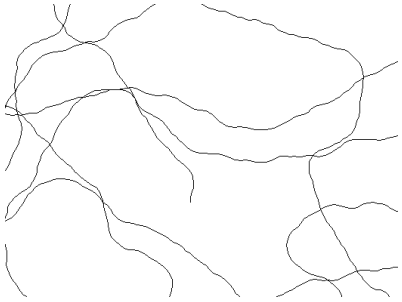
Náhodný pohyb želvy

A jak by tedy vypadal náhodný pohyb želvy? Předpokládejme, že krok želvy bude v nějakém intervalu a po každém kroku se želva otočí doprava také o hodnotu z nějakého intervalu (pootočení doleva je totéž co doprava o záporný úhel).

```
? to nahodny.pohyb.1 :k1 :k2 :u1 :u2
> forward (interval :k1 :k2)
> right (interval :u1 :u2)
> nahodny.pohyb.1 :k1 :k2 :u1 :u2
> end
Nadefinoval jsi nahodny.pohyb.1
```

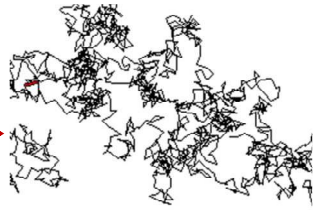
Procedura je rekurzivní, to znamená, že nikdy nedojde k ukončení pohybu želvy (pohyb ale vždy můžeme přerušit tlačítkem STOP). Nyní můžeme vyzkoušet náhodný pohyb želvy.

```
? nahodny.pohyb.1 5 8 -20 20
```



A je čas pro experimenty. Jaké parametry bychom měli zadat, aby se želva pohybovala více vpravo nebo vlevo (tak postupuje člověk, který zabloudí v neznámém terénu)?

```
? nahodny.pohyb.1 2 10 0 360
```



Jak omezit náhodný pohyb želvy

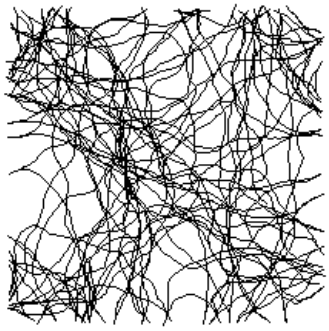
Pokusme se nyní želvě omezit oblast, ve které se může pohybovat. Uděláme to tak, že budeme kontrolovat, kdy želva překročí hranice, v tomto okamžiku ji otočíme zpět (right 180) a provedeme velký krok. Následuje procedura, která takový algoritmus realizuje. Pro testování polohy želvy později napíšeme proceduru **je.mimo.2.**

```
? to nahodny.pohyb.2 :k1 :k2 :u1 :u2
> right interval :u1 :u2
> forward interval :k1 :k2
> if je.mimo? [right 180 forward :k2]
> nahodny.pohyb.2 :k1 :k2 :u1 :u2
> end
Nadefinoval jsi nahodny.pohyb.2
```

Pro rozhodování je v jazyku LOGO k dispozici příkaz **if**, který na základě splněné podmínky provede příkazy v hranatých závorkách.
A teď k proceduře ke zjištění polohy želvy. Předpokládejme, že želvu omezíme **čtvercem o straně 200 jednotek**. Když začínáme, má želva polohu [0 0]. Nesmí se tedy dostat dále než 100 jednotek na každou stranu (resp. pokud tuto vzdálenost překročí, musí se okamžitě vrátit). K tomu ovšem potřebujeme znát aktuální polohu želvy. Tu zjistíme pomocí příkazu **pos**, který vrátí dvě čísla, totiž polohu želvy ve vodorovném a svislém směru, např. [93 -101]. První hodnotu pak získáme příkazem **first pos**, druhou příkazem **last pos**. Uvedená hodnota ovšem musí vracet hodnotu „ano“ nebo „ne“. Ty jsou reprezentovaný jmény **true** a **false**. Pokud tato jména použijeme v programu, musí začínat uvozovkami.

```
? to je.mimo?
> if first pos > 100 [output "true]
> if first pos < -100 [output "true]
> if last pos > 100 [output "true]
> if last pos < -100 [output "true]
> output "false
> end
Nadefinoval jsi je.mimo?
```

```
? nahodny.pohyb.2 1 5 -20 20
```



Třetí náhodný pohyb bude ještě v menším prostoru. Želva totiž nemůže „dovnitř" čtverce o straně 100 jednotek. Test je reprezentován procedurou **je.mimo.3**. Problém ovšem je, že želva teď musí udělat čelem vzad ve dvou různých případech, přesněji řečeno, pokud nastane první (želva jde ven z velkého čtverce) nebo druhý (želva jde dovnitř malého čtverce). Takový test realizujeme pomocí logické spojky **or**.

```
? to nahodny.pohyb.3 :k1 :k2 :u1 :u2
> right interval :u1 :u2
> forward interval :k1 :k2
> if (or je.mimo? je.mimo2?) [right 180 forward :k2]
> nahodny.pohyb3 :k1 :k2 :u1 :u2
> end
```

Test **je.mimo2?** potom naopak potřebuje, aby testoval zda je splněno více podmínek najednou (uvažme proč). K tomu slouží spojka **and**.

```
? to je.mimo2?
> if (and (first pos < 50) (first pos > -50) (last pos < 50) (last pos > -50)) [output "true]
> output "false
> end
```

```
? nahodny.pohyb.3 1 10 -20 20
```

*Poznámka: Samozřejmě bychom mohli postupovat stejně jako v proceduře je.mimo?, tj. bez použití **and**, nebo naopak.*

Poznámka: Při vyzkoušení třetího náhodného pohybu musí na začátku ležet želva v „povoleném" území. V opačném případě chvilku trvá, než se do povoleného území dopstane sama.

Úkol: Omezte želvě ještě více „životní prostor", např. několika malými obdélníky uvnitř jednoho velkého.

Ukončení rekurze

Prozatím jsme rekurzivní procedury psali tak, že jsme nepředpokládali jejich samostatné ukončení (běh jsme přerušili tlačítkem **STOP**). Protože nyní známe příkaz **if,** mohli bychom

rekurzi ukončit při splnění nějaké podmínky. Tak by tomu správně mělo být vždy.

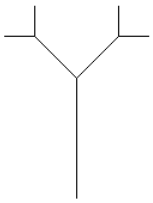
Strom

Možná to tak nevypadá, ale strom má vlastně rekurzivní tvar. Větve totiž můžeme považovat za malé samostatné stromy atd.. Napišme proceduru, která nakreslí pravidelný strom, přičemž jako parametry zadáme velikost kmene a počet rozvětvení.

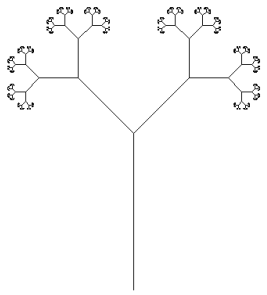
```
? to strom :kmen :pocet
> if :pocet = 0 [stop]
> forward :kmen
> left 45
> strom (:kmen / 2) (:pocet - 1)
> right 90
> strom (:kmen / 2) (:pocet - 1)
> left 45
> back :kmen
> end
```

Procedura pracuje tak, že pokud má být počet rozvětvení 0, ukončí příkazem **stop** svoji činnost. Jinak se želva pootočí doleva, nakreslí poloviční strom otočí se doprava a znovu nakreslí poloviční strom. Důležité je, aby se želva po ukončení procedury vrátila zpět na původní místo. Proč?

```
? strom 100 3
```



```
? strom 200 12
```



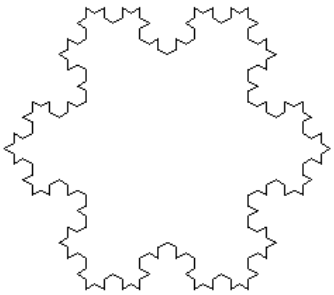
Sněhová vločka

Sněhová vločka může mít opět rekurzivní tvar (koneckonců jako mnoho věcí v přírodě, zkusme najít jiné příklady). Napišme a vyzkoušejme další procedury a pokusme se zdůvodnit, proč dělají to, co dělají.

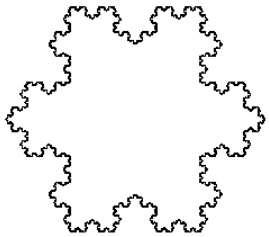
```
? to snehova.vlocka :a :pocet
> repeat 3 [strana :a :pocet right 120]
> end

? to strana :a :pocet
> if :pocet = 0 [forward :a stop]
> strana (:a/3) (:pocet-1)
> left 60
> strana (:a/3) (:pocet-1)
> right 120
> strana (:a/3) (:pocet-1)
> left 60
> strana (:a/3) (:pocet-1)
> end
```

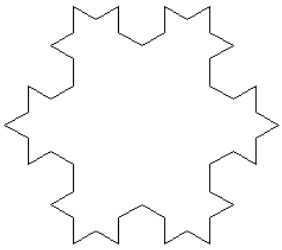
```
? snehova.vlocka 200 3
```



```
? snehova.vlocka 200 8
```



```
? snehova.vlocka 200 2
```

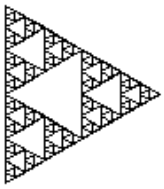


A nakonec trojúhelník

Proč se jmenuje hustý trojúhelník?

```
? to husty.trojuhelnik :a
> if :a < 5 [stop]
> repeat 3 [husty.trojuhelnik (:a/2) forward :a right 120]
> end
```

```
? husty.trojuhelnik 100
```



Pozorně prostudujte předchozí text a zkuste zodpovědět následující otázky k zopakování a upevnění učiva:

1. *Co je to rekurze?*
2. *V jakých procedurách lze rekurzi využít?*
3. *Proč je třeba „hlídat“, jestli bude rekurzivní procedura ukončena?*
4. *Jak pomocí rekurze naprogramovat vykreslení mnohoúhelníku? Jaké jsou možnosti?*
5. *Co je rekurze s návratovou hodnotou?*
6. *Co je to funkce?*
7. *K čemu slouží příkaz random?*
8. *Jak lze omezit pohyb želvy?*
9. *Jakými způsoby lze naprogramovat vykreslení stromu?*
10. *Jakými způsoby lze naprogramovat vykreslení sněhové vločky?*