Conteúdo

Introdução

Nesse manual, você encontrará um certo número de ingredientes a fim de dominar a linguagem Logo. Cada capítulo tenta abordar uma noção específica da linguagem e propõe atividades a serem realizadas quer à maneira de descoberta quer para auto-avaliação (sim, há erros cometidos propositalmente neste manual - verifique as correções disponíveis no seu final).

Com este manual não se pretende oferecer o tutorial ideal para apreender o LOGO. Satisfaz-nos propor diversas pistas, diferentes abordagens sobre certos aspectos da programação em LOGO. As atividades são de qualquer nível e, na opinião do autor original, representa um bom painel do que é capaz de se realizar com a linguagem LOGO. Espero que as explicações trazidas sejam as mais claras possíveis! Não hesitem em fazer-nos chegar seus comentários e suas críticas em relação a este tutorial. E agora comecem a iniciar-se nas alegrias da pequena tartaruga!

Explorar as primitivas básicas

Para mover a tartaruga ("tat") pela tela, usamos “primitivas” (que são comandos pré-definidos). Neste primeiro capítulo, exploraremos as primitivas básicas que permitem controlar a "tat".

Subsecções

Primitivas a usar:

1. [ ] `pf número` (`pf 50`)
   Faz a tat avançar o número de passos de tartaruga indicados por um número.
2. [ ] `pt número` (`pt 100`)
   Faz a tat recuar o número de passos de tartaruga indicados por um número.
3. [ ] `pd número` (`pd 90`)
   A tat gira para a direita tantos graus quanto indicados.
4. [ ] `pe número` (`pe 45`)
   A tat gira para a esquerda tantos graus quanto indicados.
5. [ ] `ld`
   Limpa os desenhos na tela (ld ou limpadesenho) e reinicializa a tat ao centro da tela (sua "origem").
6. [ ] `at`
   A tat fica visível na tela.
7. [ ] `dt`
   Esconde a tat (ela "desaparece"). Interessante recurso quando se deseja que um programa seja mais rápido.
8. [ ] `un`
   Levanta o lápis. A tat não deixa mais traço ao deslocar-se..
9. [ ] `ul`
   Use lápis (equivale a "baixecaneta" em outros programas Logo). A tat desenha ou escreve por onde passa.
10. [ ] `repita número (lista repita 5[pf 50 pd 45])`
    Repete as instruções contidas na lista tantas vezes quanto indicadas pelo número.

Desenhar um polígono regular

Aqui, vamos aprender a desenhar um quadrado, um triângulo equilátero, um pentágono regular etc.

Subsecções

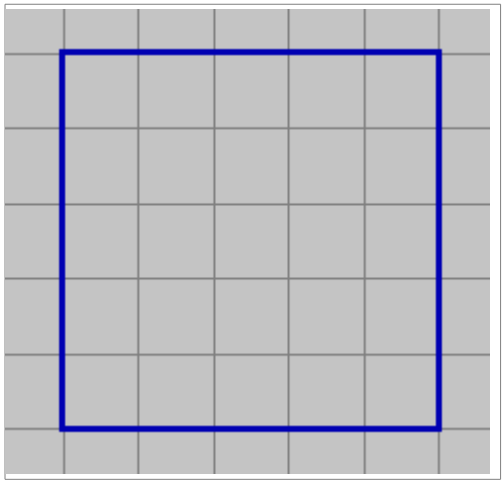
O quadrado

(Cada quadrícula na figura ao lado representa 40 passos de tartaruga.)
Para desenhar o quadrado proposto, deve-se escrever:

```
pf 200 pd 90 pf 200 pd 90 pf 200 pd 90 pf 200 pd 90
```

Uma forma mais elegante e rápida seria escrever essa mesma instrução da seguinte forma:

```
repita 4[pf 200 pd 90]
```



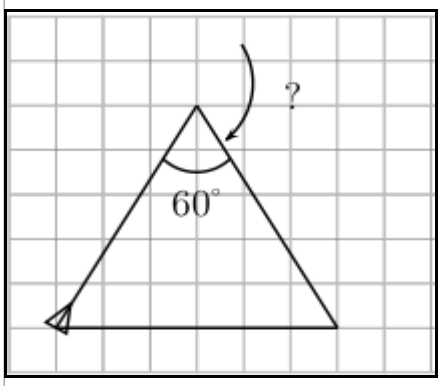
O triângulo equilátero

(Aqui, cada quadrícula representa 30 passos de tartaruga).
Vejamos como desenhar um triângulo equilátero de 150 passos de tartaruga de lado.
O comando deverá ser algo no estilo:

```
repita 3[pf 150 pd ....]
```

Resta determinar o ângulo adequado. Num triângulo equilátero, os ângulos valem todos 60°. Como a tat deve girar fora do triângulo. O ângulo valerá 180-60=120°. Então o comando deverá ser:

```
repita 3[pf 150 pd 120]
```



O hexágono

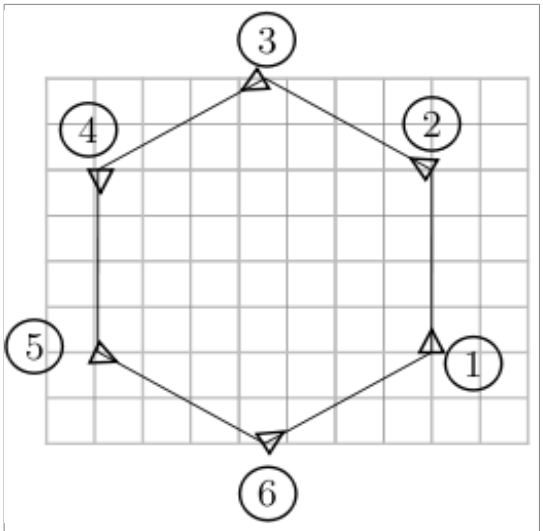
Dessa vez, cada quadrícula vale 20 passos de tartaruga.

```
repita 6[pf 80 pd ....]
```

Nota-se que, ao deslocar-se, a tat dá uma volta completa sobre ela mesma. Esta rotação de 360° é feita em 6 etapas. Por conseguinte, cada vez, gira $\frac{360}{6} = 60°$.

O comando deverá ser:

```
repita 6[pf 80 pd 60]
```



Desenhar um polígono regular qualquer

Mais uma vez, seguindo o pequeno raciocínio anterior, deve-se notar que para desenhar um polígono de n lados, o ângulo será obtido dividindo 360 por n . Por exemplo:

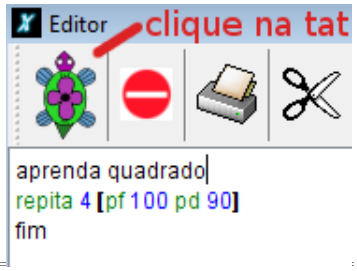
- Para desenhar um pentágono regular de lado 100:
  `repita 5[pf 100 pd 72] (360:5=72)`
- Para desenhar um enágono regular (9 lados) de lado 20:
  `repita 9[pf 20 pd 40] (360:9=40)`
- Para desenhar um..., digamos, 360-gono regular de lado 2: (este aqui é bem parecido com um círculo, não?)
  `repita 360[pf 2 pd 1]`
- Para desenhar um heptágono de lado 120:

  ```
  repita 7[pf 120 pd 360/7]
  ```

Registrar um procedimento

Para evitar digitar repetidas instruções toda vez que quiser desenhar um quadrado, um triângulo, etc. pode-se definir instruções pessoais chamadas procedimentos . Um procedimento começa pela palavra `aprenda` e termina com a palavra `fim`. Ao abrir o editor, digitaríamos, por exemplo

```
aprenda quadrado
repita 4[pf 100 pd 90]
fim
```

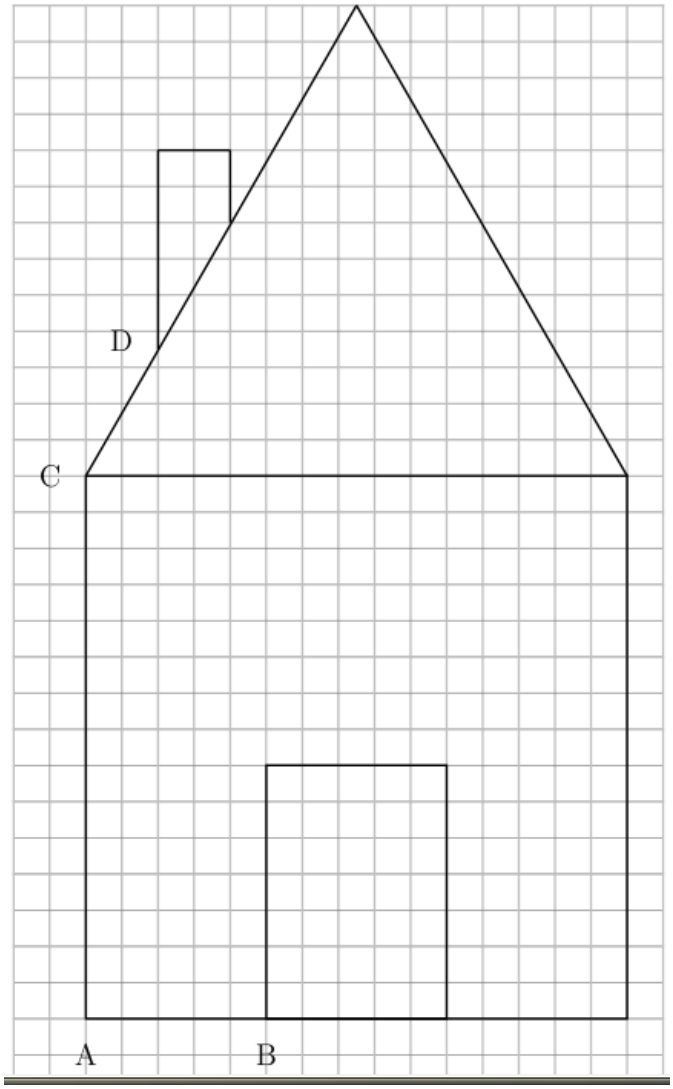


Em seguida, clica-se na figura da tartaruga para fechar o editor de modo a registrar ("gravar") as modificações. Agora, cada vez que digitar `quadrado`, um quadrado será desenhado na tela!

Atividade ...

Um pequeno quadrado com 10 passos de tartaruga.
Sua tarefa será fazer o desenho abaixo. Para isso, deve-se definir oito procedimentos:

- Um procedimento `quadrado` que desenhe o quadrado da base da casa.
- Um procedimento `tri` que desenhe um triângulo equilátero (o teto da casa).
- Um procedimento `porta` que desenhe um retângulo que represente a porta.
- Um procedimento `chamine` que desenhe uma chaminé
- Um procedimento `dep1` que permita a tat ir da posição A até a posição B.
- Um procedimento `dep2` que faça a tat ir da posição B até a posição C.
- Um procedimento `dep3` que faça a tat ir da posição C até a posição D. (Atenção, talvez seja necessário levantar o lápis da tat...)
- Um procedimento `casa` que faça desenhar a casa "chamando" os procedimentos anteriores.



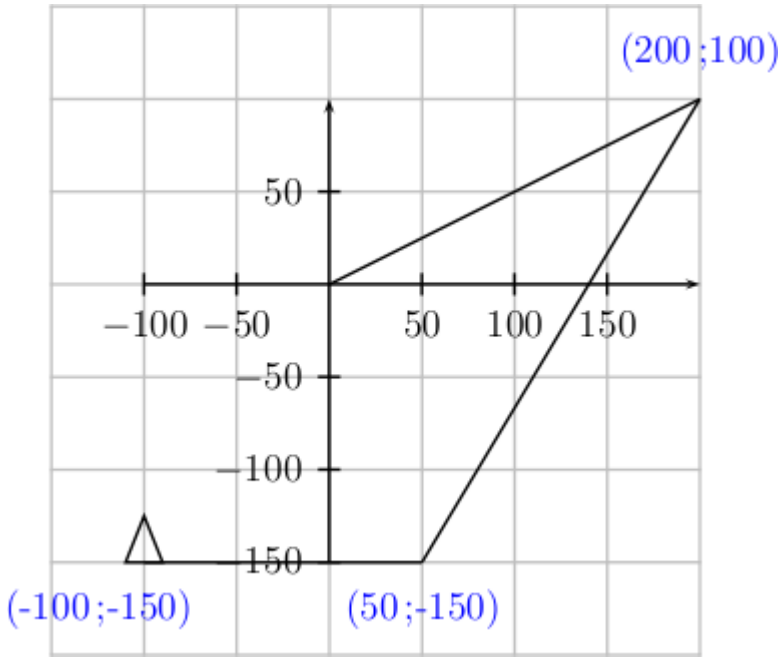
Subsecções

Apresentação

Nesse capítulo, exploraremos a primitiva `mudepos`. A área de desenho possui um marcador cuja origem está situada no centro da tela. Pode-se assim atingir cada um dos pontos da área de desenho através das suas coordenadas.

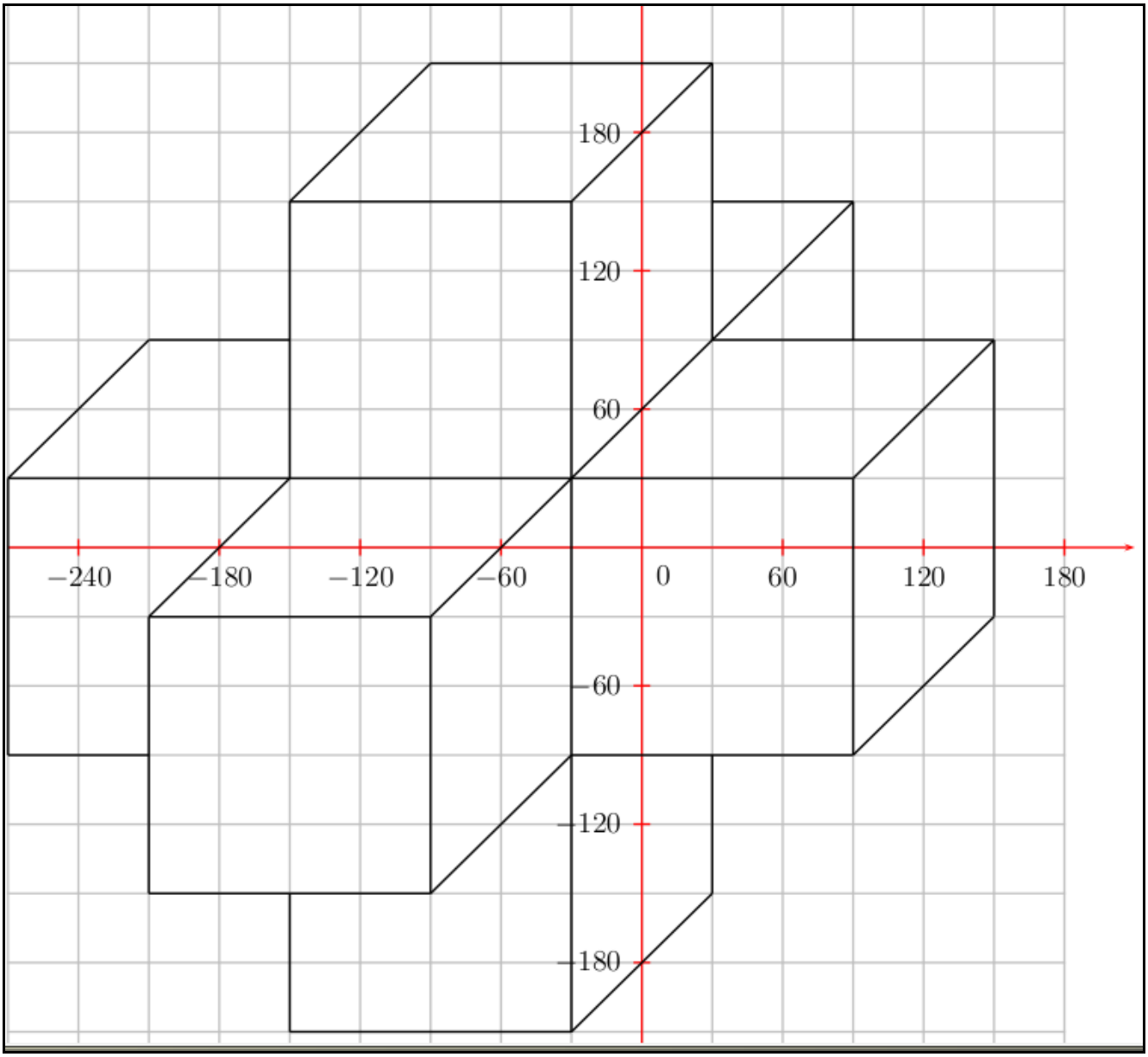
`mudepos lista` `mudepos [100 -250]`
Deslocará a tat até o ponto cujas coordenadas são definidas na lista.

Um pequeno exemplo de uso:
```
ld mudepos [200 100] mudepos [50 -150] mudepos [-100 -150]
```



Atividade:

Nesta atividade deverá ser desenhada a figura a seguir. Será permitido usar apenas as primitivas: `mudepos`, `ld`, `un`, `ul`.



As variáveis

Subsecções

O papel das variáveis

Por vezes, deseja-se desenhar uma mesma forma mas com dimensões diferentes. Por exemplo, se desejássemos desenhar um quadrado de lado 100, um quadrado de lado 200 e um quadrado de lado 50; definiríamos três procedimentos diferentes que correspondessem a cada um destes quadrados. Mas não seria mais simples definir só um procedimento ao qual passaria-se em parâmetro o comprimento do lado desejado? Por exemplo, `quadrado 200` para desenhar o quadrado de lado 200, `quadrado 100` para desenhar o quadrado de lado 100, etc. É justamente nesses casos que fazemos uso de variáveis.

Exemplos práticos

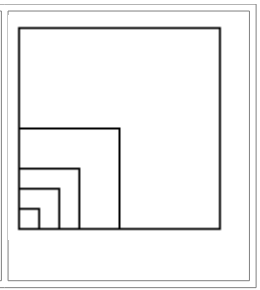
Para desenhar um quadrado de lado 100, usamos:

```
aprenda quadrado
repita 4[pf 100 pd 90]
fim
```

Vamos alterar este procedimento para que receba um parâmetro (ele diga igualmente argumento) indicando a medida do lado do quadrado a ser desenhado.
Uma variável será precedida do símbolo ":" (*dois pontos*). Se quisermos indicar ao procedimento `quadrado` que ele dependerá da variável `:c`, acrescentaremos `:c` ao fim da linha de definição. Desse modo, a tat avançará `:c` passos de tartaruga em vez dos 100 passos definidos no procedimentos acima. O procedimento ficará assim:

```
aprenda quadrado :c
repita 4 [pf :c pd 90]
fim
```

Assim, escrevendo:
`quadrado 100 quadrado 50 quadrado 30 quadrado 20 quadrado 10`
Obteremos a figura ao lado.



Desenhar um retângulo de altura e largura determinadas

Vamos aqui definir um procedimento de nome `retangulo` que dependerá de duas variáveis.
Por exemplo, `rec 200 100` desenhará um retângulo de altura 200 e largura 100. Obteremos:

```
aprenda retangulo :alt :lar
repita 2 [pf :alt pd 90 pf :lar pd 90]
fim
```

Experimentemos então:

```
retangulo 200 100 retangulo 100 300 retangulo 50 150 retangulo 1 20 retangulo 100 2
```

Certamente, se fornecermos apenas um argumento ao procedimento `retangulo`, o interpretador devolverá uma mensagem de erro informando que o procedimento precisa de mais um argumento.

Desenhar uma forma com diferentes dimensões

Retomando o exemplo da casa no Capítulo 1, vejamos como alterar o código de modo que aprenda a desenhá-la em ordem com quaisquer dimensões.

O objetivo é colocar um argumento no procedimento `casa` para que de acordo com o parámetro, a casa ou seja maior ou seja menor. Desejamos que `casa 10` desenhe a casa do Capítulo 1.
`casa 5` desenhará uma casa na escala 0,5.
`casa 20` desenhará uma casa com dimensões duas vezes maior, etc.

A noção de proporcionalidade é certamente subjacente. Sobre o desenho, do capítulo 1, um quadrado representa 10. O procedimento `quadrado` era assim:

```
aprenda quadrado
repita 4[pf 150 pd 90]
fim
```

Agora deverá ficar assim:

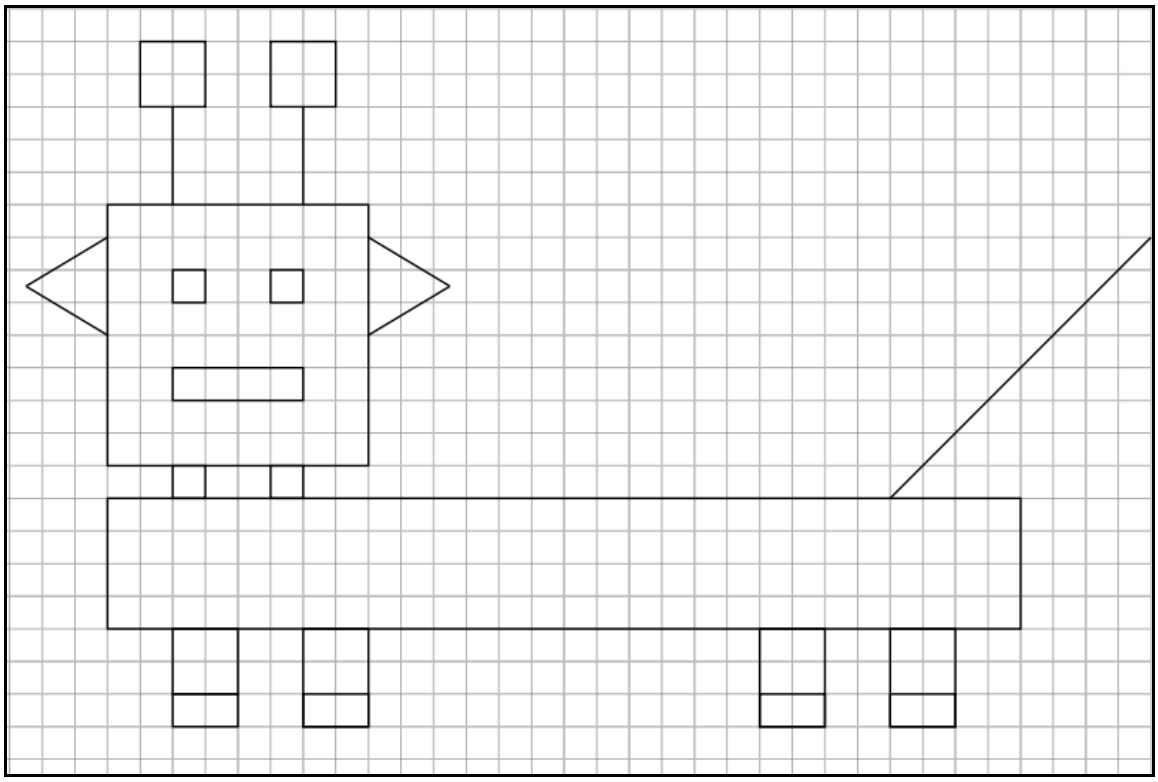
```
aprenda quadrado :c
repita 4[pf 15*:c pd 90]
fim
```

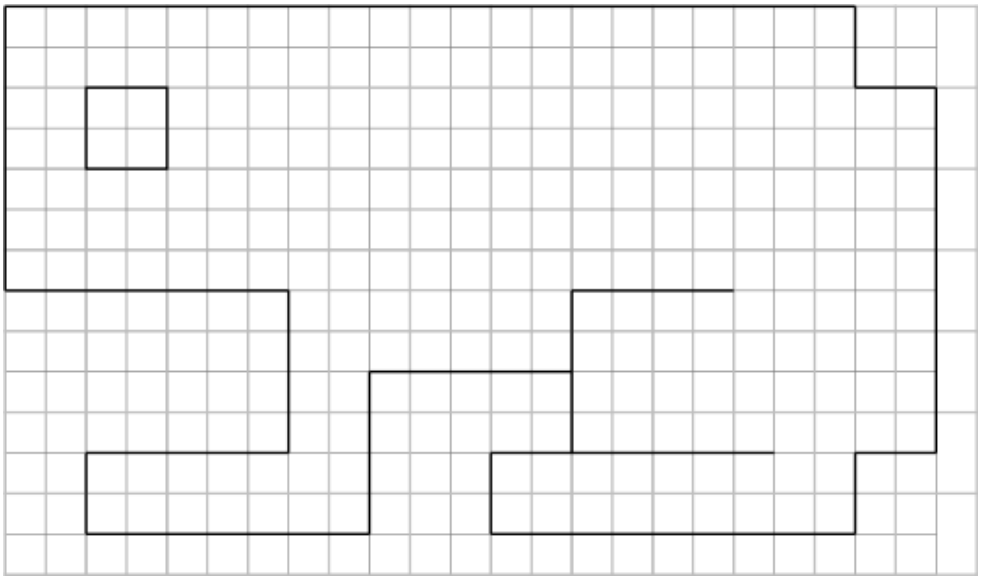
Assim, ao escrever `quadrado 10`, o quadrado terá lado 15 x 10 =150. As proporções serão bem respeitadas! Com efeito, apercebe-se-se que vai exatamente ser necessário retomar todos os procedimentos e alterar os comprimentos de deslocamento como segue.
`70` tornar-se-á `7*:c`
`pf 45` tornar-se-á `pf 4.5*:c`
etc

De modo mais simples, bastará contar o número de quadrículas para cada comprimento! Assim, teremos:

```
aprenda quadrado :c
repita 4 [pf 15*:c pd 90]
fim
aprenda tri :c
repita 3 [pf 15*:c pd 120]
fim
aprenda porta :c
repita 2 [pf 7*:c pd 90 pf 5*:c pd 90]
fim
aprenda chamine :c
pf 5.5*:c pd 90 pf 2*:c pd 90 pf 2*:c
fim
aprenda dep1 :c
pd 90 pf 5*:c pe 90
fim
aprenda dep2 :c
pe 90 pf 5*:c pd 90 pf 15*:c pd 30
fim
aprenda dep3 :c
lc pd 60 pf 2*:c pe 90 pf 3.5*:c bc
fim
aprenda casa :c
quadrado :c dep1 :c porta :c dep2 :c tri :c dep3 :c chamine :c
fim
```

Atividade
Realizar os desenhos abaixo com variáveis de modo que se possa obtê-los com diversas dimensões.





A recursividade

Diz-se que um procedimento é **recursivo** se chamar ele mesmo. Vejamos alguns exemplos que utilizam esta propriedade.

Subsecções

Com a área de desenho.

Subsecções

Primeiro exemplo:

Digite o seguinte procedimento no editor:

`aprenda ex1`
`pd 1`
`ex1`
`fim`
	Este procedimento é recursivo dado que o comando `ex1` é executado na última linha. Ao executá-lo, constata-se que a tartaruga não cessa de girar sobre si. Para interromper o programa, somos obrigados a usar o botão PARE.

Três novas primitivas:

`espere número` (`espere 60`)
Bloqueia o programa durante 60 avos do número indicado. Por exemplo, `espere 120` interromperá o procedimento durante dois segundos.

`useborracha, ub` (`useborracha`)
A tartaruga desloca-se apagando tudo por onde passa.

`lápispinta,lp` (`lápispinta`)
Passa ao modo padrão: a tartaruga deixa um traço ao deslocar-se.

Segundo exemplo:

`aprenda ex2`
`pf 200 useborracha espere 60`
`pt 200 lápispinta pd 6`
`ex2`
`fim`
	Tente adivinhar o que fará este programa.
	
	Executar o comando `ld ex2`
	Um bonito ponteiro dos segundos!

Com a área de texto

Subsecções

Um primeiro exemplo:

Digite sucessivamente `mostre "bem-vindo`, `mo "bem-vindo`, `mo [Escrevo o que quiser]`
Imagino que você já domine a primitiva `mostre` ou `mo`.
Não esquecer o " quando se quer escrever uma palavra. Melhor ainda escrever entre colchetes ([ ]) pois eles aceitarão várias palavras de uma só vez.

`aprenda ex3 :n`
`mostre :n`
`ex3 :n+1`
`fim`
	Executar o comando `ex3 0`
	(Interrompa com o botão PARE)
	Feitas as mudanças necessárias neste programa para que os números apareçam de dois em dois.
	
	Desejamos apresentar todos os números superiores a 100 que sejam múltiplos de cinco. Que fazer no programa? Que escrever para rodá-lo?

Realizar um teste de saída

Digite os seguintes comandos:

```
se 2+1=3 [mostre [isto é verdadeiro]]
se 2+1=4 [mostre [isto é verdadeiro]][mostre [o cálculo é falso]]
se 2+5=7 [mo "verd][mo "falso]
```

Se estiver em dúvida quanto à sintaxe da primitiva `se`, consulte o manual de referência do xLogo.

`aprenda ex3 :n`
`se :n=100 [pare]`
`mostre :n`
`ex3 :n+1`
`fim`
	Executar o comando `ex3 0`
	Faça as mudanças necessárias neste programa para indicar os números compreendidos entre 55 e 350 que sejam múltiplos de 11.

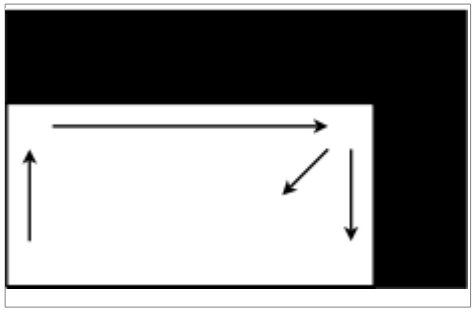
Algumas técnicas de preenchimento

Nesta lição, veremos como proceder para preencher um retângulo de comprimento e altura determinadas. Escolheremos nos exemplos seguintes um retângulo de 100 por 200.

Subsecções

Primeira abordagem

Se desejarmos, por exemplo, desenhar um retângulo preenchido de 100 por 200, uma primeira idéia pode ser desenhar o retângulo de 100 por 200, depois desenhar um retângulo de 99 por 199, depois um retângulo de 98 por 198... até o retângulo ser preenchido inteiramente.
Comecemos por definir um retângulo de comprimento e altura que dependa de duas variáveis.



```
aprenda rec :alt :lar
repita 2[pf :alt pd 90 pf :lar pd 90]
fim
```

Para preencher o nosso grande retângulo, vai-se por conseguinte executar:
`rec 100 200 rec 99 199 rec 98 198 ..... rec 1 101`
Definindo então um procedimento retângulo dedicado a desenhar este retângulo preenchido.

```
aprenda retangulo :alt :lar
rec :alt :lar
retangulo :alt-1 :lar-1
fim
```

Ao testarmos `retângulo 100 200` percebe-se que há um problema: o procedimento não se para quando o retângulo é preenchido, ela continua a desenhar retângulos! Vai-se por conseguinte acrescentar um teste que permite detectar se o comprimento ou a amplitude é igual a 0. Para isso, pede-se ao programa que se interrompa com a primitiva `pare`.

```
aprenda retangulo :alt :lar
se ou :alt=0 :lar=0 [pare]
rec :alt :lar
retangulo :alt-1 :lar-1
fim
```

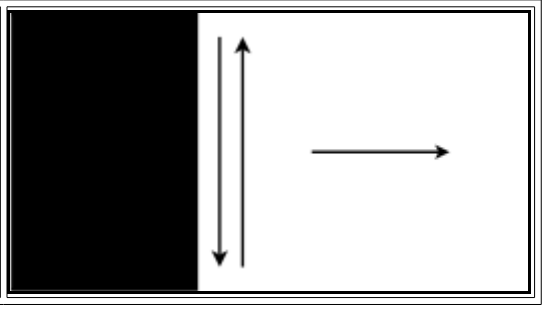
Nota: em vez de utilizar a primitiva `ou`, podemos usar o símbolo | . Assim:

```
se :alt=0 | :lar=0 [pare]
```

Segunda abordagem

A idéia aqui é começar por avançar 100 passos e recuar esses mesmos 100 passos para depois avançar um passo apara a direita. Esse movimento será repetido tantas vezes até se obter um retângulo totalmente preenchido. Se altura do retângulo for dada pela variável `:alt`, a tat terá que fazer o seguinte movimento básico:

```
pf :alt pt :alt pd 90 pf 1 pe 90
```



Este movimento deverá se repetir `:lar` vezes. O procedimento final fica assim:

```
aprenda retangulo :alt :lar
pf :alt pt :alt
repita :lar-1 [ pd 90 pf 1 pe 90 pf :alt pt :alt]
fim
```

Nota: Temos no exemplo que a tat desenha um traço vertical e em seguida fará um pequeno traço (de um passo) para o lado toda vez que chega na base do retângulo.

Uma outra forma seria utilizar recursão com um teste para interrompê-la.

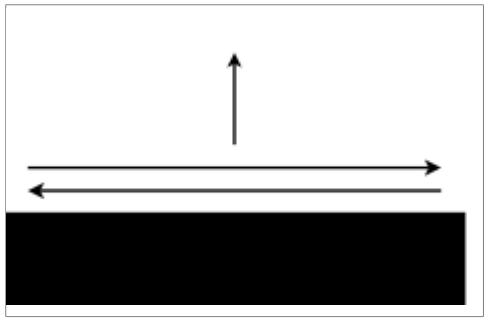
```
aprenda retangulo :alt :lar
se :lar=0 [pare]
pf :alt pt :alt
se não :lar=1 [pd 90 pf 1 pe 90]
retangulo :alt :lar-1
fim
```

Nota: A cada traço vertical desenhado, a variável `:lar` é descontada em um unidade. Assim, ao atingir o valor 0, o retângulo estará completo.

Terceira abordagem

Efetua-se o mesmo movimento de anteriormente mas desenhando sucessivamente os traços horizontais.

```
aprenda retangulo :alt :lar
pd 90 pf :lar pt :lar
repita :alt-1 [ pe 90 pf 1 pd 90 pf :lar pt :lar]
fim
```

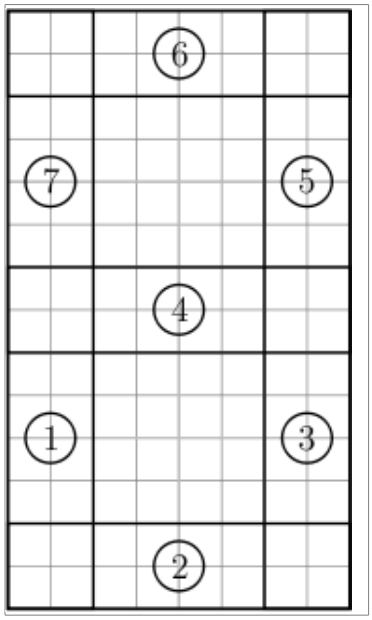


Caracteres de calculadora

Esta atividade é baseada no fato de que todos os números de calculadora podem ser obtidos através da combinação de retângulos:

Por exemplo, para desenhar um 4, acenderá os retângulos 3,4,5,7.
Para desenhar um 8, acenderá os retângulos 1,2,3,4,5,6,7.
Para desenhar um 3, acenderá os retângulos 2,3,4,5,6.

(na figura ao lado, cada quadrícula corresponde a 20 passos de tartaruga)



Subsecções

O programa

Teremos necessidade do retângulo preenchido como vimos:

```
aprenda rec :alt :lar
se :alt=0 |:lar=0[pare]
repita 2[pf :alt pd 90 pf :lar pd 90]
rec :alt-1 :lar-1
fim
```

Consideraremos aqui que a tat parte do canto inferior esquerdo. Vamos definir um procedimento chamado `numero` admitindo 7 argumentos `:a`, `:b`, `:c`, `:d`, `:e`, `:f`, `:g`. Quando `:a` valer 1, desenhará o retângulo 1. Se `:a` valer 0, ele nada desenhará. Eis aí o princípio.

Escrevemos assim o procedimento:

```
aprenda numero :a :b :c :d :e :f :g
# Desenha o retângulo 1
se :a=1 [rec 160 40]
# Desenha o retângulo 2
se :b=1 [rec 40 160]
un pd 90 pf 120 pe 90 ul
# Desenha o retângulo 3
se :c=1 [rec 160 40]
un pf 120 ul
# Desenha o retângulo 5
se :e=1 [rec 160 40]
# Desenha o retângulo 4
pe 90 un pt 40 ul
se :d=1 [rec 160 40]
# Desenha o retângulo 6
pd 90 un pf 120 pe 90 ul
se :f=1 [rec 160 40]
# Desenha o retângulo 7
un pf 120 pe 90 pt 40 ul
se :g=1 [rec 160 40]
fim
```

Criação de uma pequena animação

Vamos aqui simular uma contagem regressiva indicando sucessivamente os números de 9 até 0.

```
aprenda regressiva
ld dt numero 0 1 1 1 1 1 1 espere 60
ld dt numero 1 1 1 1 1 1 1 espere 60
ld dt numero 0 0 1 0 1 1 0 espere 60
ld dt numero 1 1 1 1 0 1 1 espere 60
ld dt numero 0 1 1 1 0 1 1 espere 60
ld dt numero 0 0 1 1 1 0 1 espere 60
ld dt numero 0 1 1 1 1 1 0 espere 60
ld dt numero 1 1 0 1 1 1 0 espere 60
ld dt numero 0 0 1 0 1 0 0 espere 60
ld dt numero 1 1 1 0 1 1 1 espere 60
fim
```

Pequeno problema: esse procedimento tem um efeito um tanto desagradável durante a criação de cada número. Para harmonizá-lo utilizaremos as primitivas `animado`, `pareanimado` e `veranimado`. `animado` exigia os argumentos `verd` ou `falso em versões anteriores à 0.9.90`. A partir da versão 0.9.90, no lugar de *animado verd* usamos apenas *animado*. No lugar de *animado falso*, usamos *pareanimado*.

- Se for igual a `falso`, estaria no modo padrão de exibição.
- Se for igual a `verd`, passaria ao modo animado. A tartaruga desenhará escondida, ou seja, usará a memória. A imagem só será apresentada na tela quando pedirmos ajuda da primitiva `veranimado`.

Para isso, modificamos o procedimento acima ficando assim:

```
aprenda regressiva
# passar ao modo animado
animado
ld dt numero 0 1 1 1 1 1 1 veranimado espere 60
ld dt numero 1 1 1 1 1 1 1 veranimado espere 60
ld dt numero 0 0 1 0 1 1 0 veranimado espere 60
ld dt numero 1 1 1 1 0 1 1 veranimado espere 60
ld dt numero 0 1 1 1 0 1 1 veranimado espere 60
ld dt numero 0 0 1 1 1 0 1 veranimado espere 60
ld dt numero 0 1 1 1 1 1 0 veranimado espere 60
ld dt numero 1 1 0 1 1 1 0 veranimado espere 60
ld dt numero 0 0 1 0 1 0 0 veranimado espere 60
ld dt numero 1 1 1 0 1 1 1 veranimado espere 60
# voltar ao modo padrão
pareanimado
fim
```

Explorar listas e variáveis.

Subsecções

Interagir com o usuário

Vamos realizar um pequeno procedimento que pede ao utilizador o seu sobrenome, o seu nome e a sua idade. Ao fim do questionário, o procedimento recapitulará as respostas:

```
Seu nome é:........
Seu sobrenome é: .......
Sua idade é: .......
Se você é demenor ou de maior
```

Para isso, usaremos as seuintes priomitivas:

- `leia`: `leia [Qual é a sua idade? ] "a`
  Cartaz uma caixa de diálogo que tem por título o texto contido na lista (aqui, Qual é a sua idade?). A resposta dada pelo utilizador é memorizada sob a forma de uma lista na variável `:a`. Por exemplo, se escrevermos 20, a variável `:a` conterá `[20]`
- `atr`: `atr "a 30`
  Atribui o valor 30 à variável `:a`
- `sentença`, `sn`: `sentença [30 k] "a`
  Acrescenta um valor numa lista. Se este valor for uma lista, juntará as duas listas em uma.
  ```
  sentença [30 k] "a ---> [30 k a]
  sentença [1 2 3] 4 ---> [1 2 3 4]
  sentença [1 2 3] [4 5 6] ---> [1 2 3 4 5 6]
  ```
- `pri`: Devolve o primeiro elemento contido numa lista:
  ```
  mostre pri [4 5 6] ---> 4
  mostre pri [comentário aqui vai 8] ----> comentário
  ```

Obtemos o seguinte código:

```
aprenda questao
leia [Qual é a sua idade?] "anos
# :anos contém então uma lista um elemento,
# podemos usar esse elemento armazenado em :anos
atr "anos pri :anos
leia [Qual é o seu sobrenome?] "sobrenome
leia [Qual é o seu nome?] "nome
mostre sentença [Seu sobrenome é: ] :sobrenome
mostre sentença [Seu nome é: ] :nome
mostre sentença [Sua idade é: ] :anos
se ou :anos>18 :anos=18 [mostre [Você é de maior]] [mostre [Você é de menor]]
fim
```

Programar um pequeno jogo.

Nosso objetivo a seguir é criar o seguinte jogo:

O programa escolhe um número entre 0 e 32 e memoriza-o. Uma caixa de diálogo abre-se e pede ao usuário que retorne um número. Se o número proposto for igual ao número memorizado, escreverá acertou na zona de texto. Caso contrário, o programa indicará que o número memorizado é menor ou maor que o número proposto pelo usuário e reabrirá nova caixa de diálogo. O programa termina-se quando o usuário encontrar o número memorizado.

Teremos necessidade de usar as seguintes primitivas:
`sorteie`: `sorteie 8` sorteia um número entre 0 e 8 apenas.
`sorteie 20` escolhe (sorteia) um número entre 0 e 19.

Coloque estas regras a serem cumpridas pelo jogo:

- O número memorizado pelo computador será guardado na variável de nome `número`.
- A caixa de diálogo deverá apresentar a mensagem: Escreva um número: .
- O número proposto pelo usário será guardado na variável de nome `palpite`.
- O procedimento que permite iniciar o jogo será chamado `jogo`.

Alguns ajustes possíveis:

- Registrar o número de tentativas.
- O número deverá ser entre 0 e 2000.
- Verificar se o número digitado pelo usuário é um número. Para tal, utiliza-se a primitiva `énúmero?`.
  Exemplos:

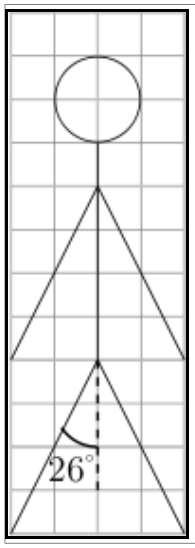
  `número? 8` é verd.

  `número? [5 6 7]` é falso. ([5 6 7] é uma lista e não um número)

  `número? "abcde` é falso. ("abcde é uma palavra e não um número)

Uma animação: o boneco que cresce

Em primeiro lugar, vamos definir um procedimento `boneco` que desenhe o homenzinho com um tamanho fornecido por nós.




```
aprenda boneco :c
pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c*2
pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c/2
pe 90 repita 180[pf :c/40 pd 2] pd 90
fim
```

Vamos agora criar uma animação que dá a ilusão de que o homenzinho cresce pouco a pouco. Para isso, vamos desenhar `boneco 1` depois `boneco 2` `boneco 3` ... até `boneco 75`. Entre cada desenho, a tela será limpa. Precisamos dos dois procedimentos seguintes:

```
aprenda boneco :c
se :c=75[pare]
pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c*2
pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c/2
pe 90 repita 180[pf :c/40 pd 2] pd 90
ld dt boneco :c+1
fim
```

```
aprenda iniciar
ld dt
boneco 0
fim
```

Enfim, para harmonizar o conjunto, nos serviremos do modo animado e da primitiva `veranimado`.

```
aprenda boneco :c
se :c=75[pare]
ld dt pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c*2
pe 154 pf 2.2*:c pt :c*2.2
pe 52 pf 2.2*:c pt :c*2.2
pe 154 pf :c/2
pe 90 repita 180[pf :c/40 pd 2] pd 90
veranimado
boneco :c+1
fim
```

Em versões anteriores ao xLogo 0.9.90:

```
aprenda iniciar
dt animado verd
boneco 0
animado falso
fim
```

Para quem usa o xLogo 0.9.90 ou posterior:

```
aprenda iniciar
dt animado
boneco 0
pareanimado
fim
```

Atividade com números primos entre si.

ALERTA: São necessárias algumas noções de matemática para compreender esse capítulo.

Subsecções

Noção de MDC (máximo divisor comum)

Dado dois números inteiros, chamamos MDC o máximo divisor comum de ambos.

- Por exemplo, 42 e 28 têm 14 como MDC (14 é o maior número que divide 28 e 42)
- 25 e 55 têm 5 como MDC.
- 42 e 23 têm 1 como MDC.

Quando dois números têm 1 como MDC, diz-se que são primos entre si. Assim sobre o exemplo precedente, 42 e 23 são primos entre si. Isso significa que não há nenhum divisor comum exceto 1 (que, certamente, divide qualquer um!).

Algoritmo de Euclides

Para determinar o MDC de dois números, usaremos um método chamado algoritmo de Euclides: (Não vamos demonstrar a validade desse algortimo, apenas admitiremos que ele funciona)

Eis o princípio: Dadas dois inteiros positivos a e b, começa-se por testar b é nulo. Se for, então o MCD é igual à a. Se não for, calcula-se r, o resto da divisão de a por b. Substitui-se a por b, e b por r, e recomeça-se o método.
Calcular, por exemplo, o MDC de 2160 e 888 por este algoritmo se dará nas seguintes etapas:

a	*b*	*r*
2160	*888*	*384*
888	*384*	*120*
384	*120*	*24*
120	*24*	*0*
24	*0*	

O MDC de 2160 e 888 é 24. Ele é o maior número inteiro que divide aqueles dois outros.
(De fato 2160 = 24 x 90 e 888 = 24 x 37)
O MDC é com efeito o último resto não nulo.

Calcular MDC em Logo

Um pequeno algoritmo recursivo permite calcular o MDC de dois números `:a` e `:b`

```
aprenda MDC :a :b
se (resto :a :b)=0 [saída :b][saída MDC :b resto :a :b]
fim
mostre MDC 2160 888 ---> 24
```

Nota: É importante colocar `resto :a :b` entre parênteses, senão o xLogo se confundiria e tentaria avaliar `:b = 0`. Para evitar esse problema de usar ou não parênteses, poderíamos ter escrito: `se 0=resto :a :b`

Calcular uma aproximação de π

Com efeito, um resultado conhecido de teoria dos números mostra que a probabilidade de dois números tomados ao acaso sejam primos entre si é de $6/\pi^2 \approx 0{,}6079$. Para testar a veradicidade desse resultado faremos:

- Tomar dois números ao acaso entre 0 e 1.000.000.
- Calcular seu MDC
- Se o MDC for 1, acrescentar "1" a variável `contar`.
- Repetir o processo 1000 vezes
- A frequência dos pares de números primos entre si obter-se-á dividindo a variável `contar` por 1000 (o número de tentativas).

```
aprenda testar
# Iniciar a variável "contar" com valor 0
atr "contar 0
repita 1000 [
se (MDC sorteie 1000000 sorteie 1000000)=1 [atr "contar :contar+1]
]
mostre [frequência:]
mostre :contar/1000
fim
```

Nota: De novo, devemos colocar `MDC sorteie 1000000 sorteie 1000000` entre parênteses para que o xLogo não se confunfa tentando interpretar como `1 000 000 = 1`. Para evitar esse problema, escreva: `se 1=MDC sorteie 1000000 sorteie 1000000`

Coloque a rodar o procedimento `testar`.

```
testar
0.609
testar
0.626
testar
0.597
```

Obtêm-se valores próximos do valor teórico de 0,6097. É notável que esta frequência é um valor aproximado de $6/\pi^2$

Seja f a frequência encontrada, então: $f \approx \frac{6}{\pi^2}$

Então $\pi^2 \approx \frac{6}{f}$ e então $\pi \approx \sqrt{\frac{6}{f}}$.

Apresso-me a acrescentar esta aproximação no meu procedimento, transformo-a no procedimento `testar`:

```
aprenda testar
# Iniciar a variável "contar" com valor 0
atr "contar 0
repita 1000 [
se 1=MDC sorteie 1000000 sorteie 1000000 [atr "contar :contar+1]
]
# Calcular a frequência
atr "f :contar/1000
# Encontrar o valor aproximado de pi
mostre sentença [aproximação de pi:] raizq (6/:f)
fim
testar
aproximação de pi: 3.164916190172819
testar
aproximação de pi: 3.1675613357997525
testar
aproximação de pi: 3.1008683647302115
```

Bom, altero o meu programa de tal forma que quando rodá-lo, informo o número de tentativas desejados. Tenho a idéia de testar com 10000 tentativas. Eis que obtenho nas minhas três primeiras tentativas:

```
aprenda testar :tentativas
# Para iniciar a variável contar com valor 0
atr "contar 0
repita :tentativas [
se 1=MDC sorteie 1000000 sorteie 1000000 [atr "contar :contar+1]
]
# Para calcular a frequência
atr "f :contar/:tentativas
# Encontrar o valor aproximado de pi
mostre sentença [aproximação de pi:] raizq (6/:f)
fim
testar 10000
aproximação de pi: 3.1300987144363774
testar 10000
aproximação de pi: 3.1517891481565017
testar 10000
aproximação de pi: 3.1416626832299914
```

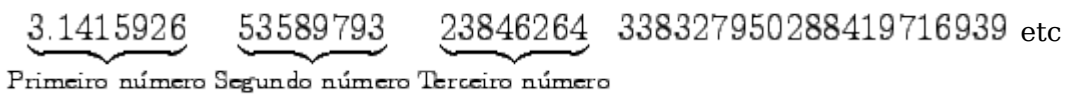
Nada mal, não?

Complicando um pouco mais: π que gera π .....

Como obter um número aleatório? Um número tomado ao sorteie entre 1 e 1 00 0000 é realmente um número aleatório? Apercebe-se muito rapidamente que nossos esforços são apenas uma aproximação do modelo ideal. Bem, é precisamente sobre a maneira de gerar o número aleatório que vamos efetuar algumas mudanças... Não vamos mais utilizar a primitiva `sorteie` mas usar a sequência de decimais de π . Deixe-me explicar: as decimaies de π sempre intrigou os matemáticos pela sua falta de irregularidade, os números de 0 para 9 parecem aparecer em quantidade mais ou menos iguais e de maneira aleatória. Não se pode predizer as próximas casas decimais através das precedentes. Vamos ver a seguir como gerar um número aleatório através das casas decimais de π . Em primeiro lugar, deveremos encontrar as primeiras casas decimais de pi (por exemplo, um milhão).

- Existem pequenos programas que o fazem muito bem. Eu aconselho PiFast para Windows e ScnhellPi para Linux.
- Você também pode visitar este endereço e efetuar um "copiar & colar" num arquivo texto: http://3.14159265358979323846264338327950288419716939937510582097494459 2.com/
- Copiar esse arquivo no sítio do xLogo: http://xlogo.tuxfamily.org/common/millionpi.txt

Para criar os números aleatórios, pegaremos pacotes de 8 algarismos em sequência de π .
Explicação: o arquivo começa assim:



Retiro o ponto decimal (.) do 3.14 ..., que corre o risco de irritar-nos ao extrair as casas decimais.
Bem, estando tudo em seu lugar, criamos um novo procedimento chamado `sorteiepi` e modificamos de leve o procedimento `testar`

```
aprenda MDC :a :b
se (resto :a :b)=0 [saída :b][saída MDC :b resto :a :b]
fim
aprenda testar :tentativas
# Ao abrir um fluxo indicado pelo número 1 para o arquivo millionpi.txt
# (considere aqui estarmos usando o diretório corrente
# senão deve-se usar uma lista e um caminho absoluto)
abrafluxo 1 "millionpi.txt
# Guardar na variável "linha" a primeira linha do arquivo millionpi.txt
atr "linha pri leialinha 1
# On initialise la variável contar à 0
atr "contar 0
repita :tentativas [
se 1=MDC sorteiepi 8 sorteiepi 8 [atr "contar :contar+1]
]
# Calcula a frequência
atr "f :contar/:tentativas
# Encontrar o valor aproximado de pi
mostre sentença [aproximação de pi:] raizq (6/:f)
fechefluxo 1
fim
aprenda sorteiepi :n
atrlocal "número "
repita :n [
# Se lê plus caracteres sobre a linha
se 0=conte :linha [atr "linha pri leialinha 1]
# Atribui à variável "caractere" o valor do primeiro caractere da linha
atr "caractere pri :linha
# Retira este primeiro caractere da linha.
atr "linha semprimeiro :linha
atr "número pal :número :caractere
]
saída :número
fim
testar 10
aproximação de pi: 2.7386127875258306
testar 100
aproximação de pi: 2.9704426289300225
testar 1000
aproximação de pi: 3.0959109381151797
testar 10000
aproximação de pi: 3.139081837741219
```

Reencontra-se por conseguinte uma aproximação do número π com ajuda de suas próprias casas decimais!!
É ainda possível melhorar este procedimento indicando por exemplo o tempo gasto para o cálculo.
Acrescenta-se então na primeira linha do procedimento testar:

```
atr "inicio tempodec
```

Acrescenta-se bem na frente de `fechefluxo 1`:

```
mostre sentença [Tempo gasto: ] tempodec - :inicio
```

Subseções

Capítulo 2

```
aprenda quadrado
repita 4[pf 150 pd 90]
fim
aprenda tri
repita 3[pf 150 pd 120]
fim
aprenda porta
repita 2[pf 70 pd 90 pf 50 pd 90]
fim
aprenda chamine
pf 55 pd 90 pf 20 pd 90 pf 20
fim
aprenda dep1
pd 90 pf 50 pe 90
fim
aprenda dep2
pe 90 pf 50 pd 90 pf 150 pd 30
fim
aprenda dep3
un pd 60 pf 20 pe 90 pf 35 ul
fim
aprenda casa
quadrado dep1 porta dep2 tri dep3 chamine
fim
```

Capítulo 3

```
aprenda supercubo
ld un mudepos[ -30 150] ul mudepos[-150 150] mudepos[-90 210] mudepos[30 210]
mudepos[-30 150]
mudepos[-30 -210] mudepos[30 -150] mudepos[30 -90] mudepos[-30 -90] mudepos[90 -90]
mudepos[90 30]
mudepos[-270 30] mudepos[-270 -90] mudepos[-210 -90] mudepos[-210 -30] mudepos[-90
-30] mudepos[-90 -150]
mudepos[-210 -150] mudepos[-210 -30] mudepos[-150 30] mudepos[-30 30] mudepos[-90
-30] mudepos[90 150]
mudepos[30 150] mudepos[30 210] mudepos[30 90] mudepos[90 90] mudepos[90 150]
mudepos[90 90] mudepos[150 90]
mudepos[150 -30] mudepos[90 -90] mudepos[90 30] mudepos[150 90] un mudepos[-150 30]
ul mudepos[-150 150]
mudepos[-150 90] mudepos[-210 90] mudepos[-270 30] un mudepos[-90 -150] ul mudepos[-
30 -90]
un mudepos[-150 -150] ul mudepos[-150 -210] mudepos[-30 -210]
fim
```

Subseções

O robô

O primeiro desenho é composto exclusivemente de um motivo elementar à base de retângulo, quadrado e triângulo. Eis o código associado a este desenho:

```
aprenda ret :alt :lar
# desenha um retângulo de altura :alt e largura :lar
repita 2[pf :alt pd 90 pf :lar pd 90]
fim
aprenda quadrado :c
# desenha um quadrado de lado :c
repita 4[pf :c pd 90]
fim
aprenda tri :c
# desenha um triângulo equilátero de lado :c
repita 3[pf :c pd 120]
fim
aprenda pata :c
ret 2*:c 3*:c quadrado 2*:c
fim
aprenda antena :c
pf 3*:c pe 90 pf :c pd 90 quadrado 2*:c
un pt 3 *:c pd 90 pf :c pe 90 ul
fim
aprenda robo :c
ld dt
# O corpo
ret 4*:c 28* :c
# As patas
pd 90 pf 2*:c pata :c pf 4* :c pata :c pf 14*:c pata :c pf 4*:c pata :c
#O rabo
un pe 90 pf 4* :c ul pd 45 pf 11*:c pt 11 * :c pe 135
# O pescoço e a cabeça
pf 18 *:c quadrado :c pf 3*:c quadrado :c pd 90 pf :c pe 90 pf 2*:c pd 90 quadrado 8*
:c
# Orelhas
pf 4*:c pe 60 tri 3*:c un pd 150 pf 8 *:c pe 90 ul tri 3*:c
# As antenas
pf 4 *:c pe 90 pf 2*:c pd 90 antena :c pe 90 pf 4*:c pd 90 antena :c
# Os olhos
un pt 3 *:c ul quadrado :c pd 90 un pf 3*:c ul pe 90 quadrado :c
# A boca
un pt 3*:c pe 90 pf 3*:c pd 90 ul ret :c 4*:c
fim
```

A rã

```
aprenda rã :c
ld dt
pf 2 *:c pd 90 pf 5*:c pe 90 pf 4*:c pe 90 pf 7 *:c pd 90 pf 7*:c pd 90
pf 21 *:c pd 90 pf 2*:c pe 90 pf 2*:c pd 90 pf 9*:c pd 90 pf 2*:c pe 90
pf 2*:c pd 90 pf 9*:c pd 90 pf 2*:c pd 90 pf 7*:c pt 5*:c pe 90 pf 4*:c
pd 90 pf 4*:c pt 4*:c pe 90 pt 2*:c pe 90 pf 5*:c pe 90 pf 4*:c pd 90 pf 7*:c
pd 90 un pf 9*:c ul repita 4[pf 2*:c pd 90]
fim
```

Capítulo 8:

```
aprenda jogo
# Inicializa o número procurado e o número de lances
atr "número sorteie 32
atr "contar 0
insistir
fim
aprenda insistir
leia [Escreva um número] "palpite
atr "palpite pri :palpite
se énúmero? :palpite[
# Testa o número dado como palpite
se :número=:palpite[mo sn sn [Você acertou em ] :contar+1 [tentativa(s)]][
se :palpite>:número [mo [É menor]][mo [É maior]]
atr "contar :contar+1
insistir
]
]
[mostre [Por favor, você deve escrever um número!] insistir]
fim
```

Sobre esse documento ...

Descubra a linguagem LOGO em 9 lições

This document was generated using the **LaTeX**2HTML translator Version 2002-2-1 (1.71)

The command line arguments were:
latex2html `-local_icons tutoriel.tex`

The translation was initiated by Alexandre R. Soares on 2006-06-18

---

alex 2007-nov-01