Fondamenti di Informatica (Elettronici)

Da THINK JULIA – Capitolo 4 (Parte a)

21 ottobre 2020

Caso di studio: Design dell'interfaccia (a)[1]

1. Tartarughe
2. Ripetizione semplice
3. Esercizi
4. Incapsulamento
5. Generalizzazione
6. Progettazione dell'interfaccia

[1] Tratto da https://benlauwens.github.io/ThinkJulia.jl/latest/book.html, disponibile sotto Licenza 'Creative Commons Attribution-NonCommercial 3.0 Unported'.

Questo capitolo presenta un caso di studio che illustra un processo per la progettazione di funzioni che collaborano.

Introduce la grafica delle tartarughe, un modo per creare disegni da programma.

La grafica delle tartarughe non è inclusa nella libreria standard, quindi il modulo ThinkJulia deve essere aggiunto alla configurazione di Julia.

Gli esempi in questo capitolo possono essere eseguiti in un taccuino (Notebook) grafico su JuliaBox, che combina codice, testo formattato, matematica e multimedia in un unico documento (il `NoteBook`).

Section 1

Tartarughe

Moduli e pacchetti (packages)

Un modulo è un file che contiene una collezione di funzioni correlate.

Julia fornisce alcuni moduli nella sua libreria standard.

È possibile aggiungere funzionalità aggiuntive da una raccolta crescente di pacchetti (https://juliaobserver.com).

I pacchetti possono essere installati in `REPL` inserendoli in `Pkg REPL-mode` usando il tasto `]`.

```
(v1.0) pkg> add https://github.com/BenLauwens/ThinkJulia.jl
```

Questo può richiedere del tempo.

Istruzione `using`

Prima di poter utilizzare le funzioni in un modulo, dobbiamo importarlo con un'istruzione `using`:

```
julia> using ThinkJulia
julia> turtle = Turtle()
Luxor.Turtle(0.0, 0.0, true, 0.0, (0.0, 0.0, 0.0))
```

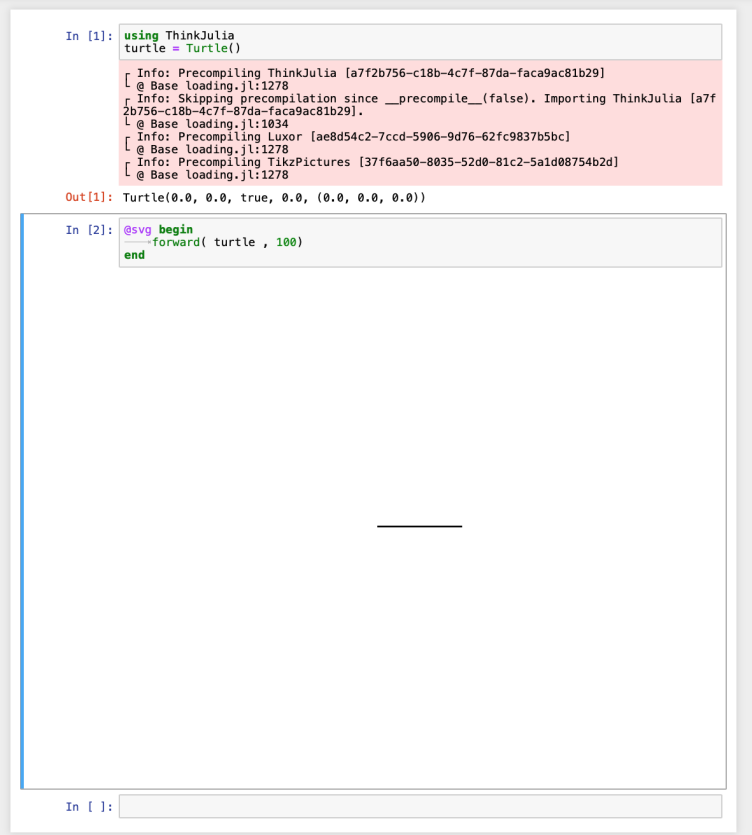
Il modulo `ThinkJulia` fornisce una funzione chiamata `Turtle` che crea un oggetto `Luxor.Turtle`, che assegniamo a una variabile chiamata (\turtle TAB).

Dopo aver creato una "tartaruga", puoi chiamare una funzione per spostarla su un disegno.

Macro svg

Ad esempio, per spostare la tartaruga in avanti:

```
@svg begin
    forward( turtle , 100)
end
```



Muovere la tartaruga in avanti La parola chiave

`@svg` esegue una macro che disegna una figura SVG. Le macro sono una funzionalità importante ma avanzata di Julia.

Gli argomenti di `forward` sono la tartaruga e una distanza in pixel, quindi la dimensione effettiva dipende dal display.

Un'altra funzione che puoi chiamare con una tartaruga come argomento è `turn` per girare. Il secondo argomento per la virata è un angolo in gradi.

penup e pendown

Inoltre, ogni "tartaruga" tiene in mano una penna, che può essere "giù" o "su";

Se la penna è "giù", la tartaruga lascia una scia quando si muove. Lo spostamento della tartaruga in avanti mostra la traccia lasciata dalla tartaruga.

Le funzioni `penup` e `pendown` stanno per "penna su" e "penna giù".

Per disegnare un angolo retto, modifica la chiamata della macro:

penup e pendown

Inoltre, ogni "tartaruga" tiene in mano una penna, che può essere "giù" o "su";

Se la penna è "giù", la tartaruga lascia una scia quando si muove. Lo spostamento della tartaruga in avanti mostra la traccia lasciata dalla tartaruga.

Le funzioni `penup` e `pendown` stanno per "penna su" e "penna giù".

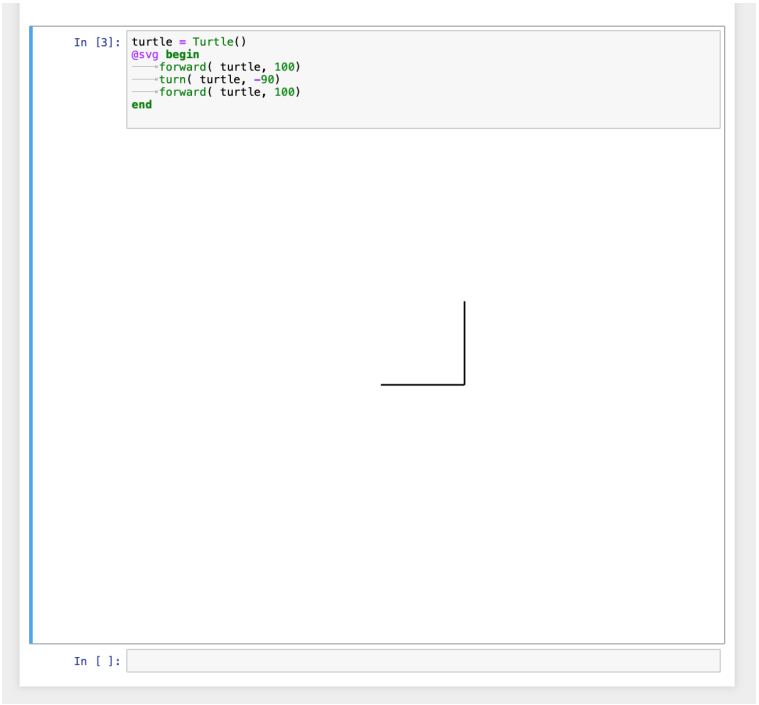
Per disegnare un angolo retto, modifica la chiamata della macro:

```
turtle = Turtle()
@svg begin
    forward( turtle, 100)
    turn( turtle, -90)
    forward( turtle, 100)
end
```

penup e pendown

Inoltre, ogni "tartaruga" tiene in mano una penna, che può essere "giù" o "su";

Se la penna è "giù", la tartaruga lascia una scia quando si muove. Lo spostamento della tartaruga in avanti mostra la traccia lasciata dalla tartaruga.



Le funzioni `penup` e `pendown` stanno per "penna su" e "penna giù".

Per disegnare un angolo retto, modifica la chiamata della macro:

```
turtle = Turtle()
@svg begin
    forward( turtle, 100)
    turn( turtle, -90)
    forward( turtle, 100)
end
```

Esercizio

Esercizio 4-1

Ora modifica la macro per disegnare un quadrato.

Non andare avanti finché non lo fai funzionare!

Section 2

Ripetizione semplice

Istruzione `for`

È probabile che tu abbia scritto qualcosa del genere:

```
turtle = Turtle()
@svg begin
    forward(turtle, 100); turn(turtle, -90)
    forward(turtle, 100); turn(turtle, -90)
    forward(turtle, 100); turn(turtle, -90)
    forward(turtle, 100)
end
```

Possiamo fare la stessa cosa in modo più conciso con un'istruzione `for`:

```
julia> for i in 1:4
           println("Hello!")
       end
Hello!
Hello!
Hello!
Hello!
```

Questo è l'uso più semplice dell'istruzione `for`. Esso dovrebbe essere sufficiente per riscrivere il programma di disegno del quadrato. Non andare avanti finché non lo fai.

Loop (ciclo) che genera un quadrato

Ecco un'istruzione `for` che disegna un quadrato:

Loop (ciclo) che genera un quadrato

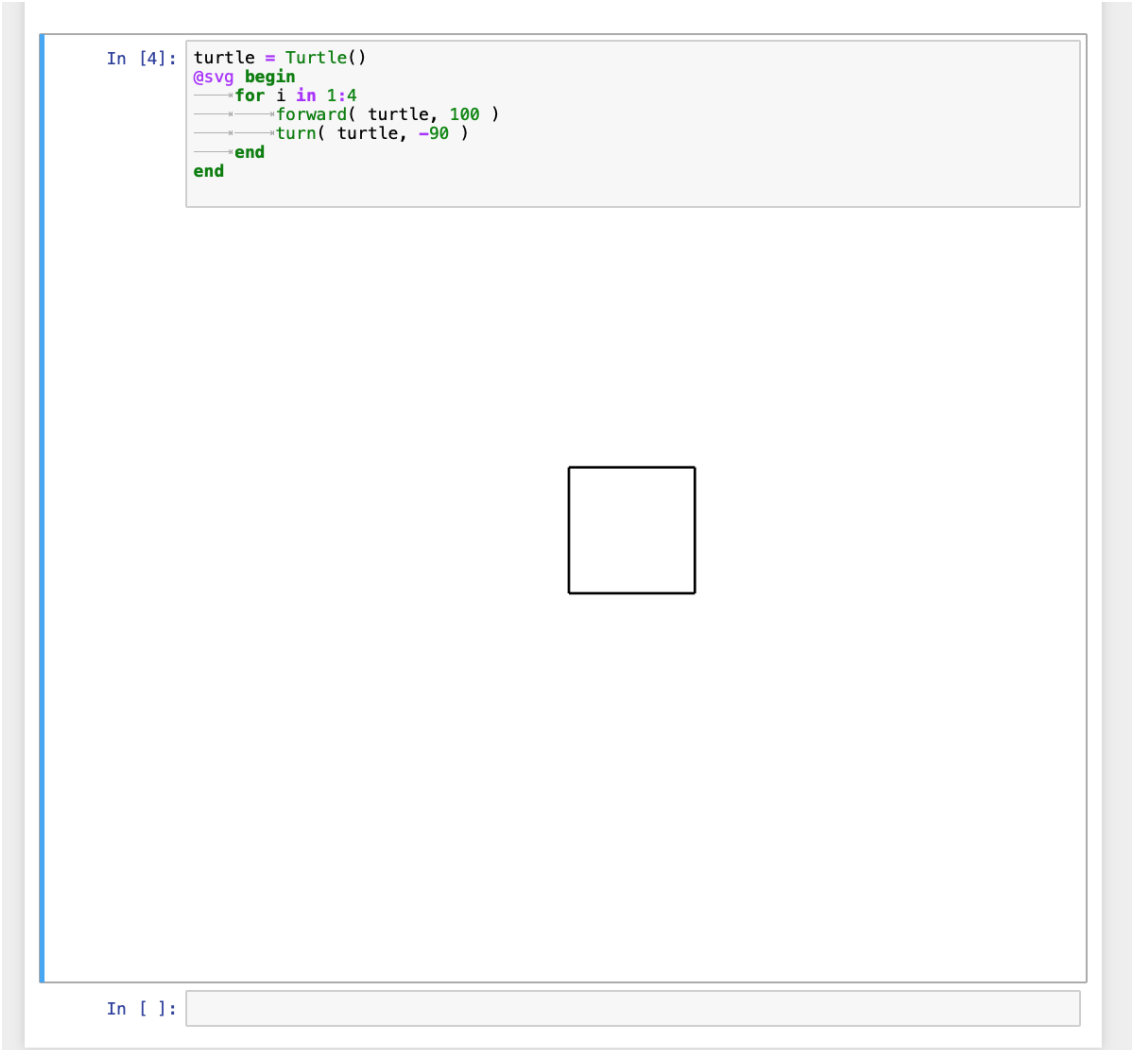
Ecco un'istruzione `for` che disegna un quadrato:

```
turtle = Turtle()
@svg begin
    for i in 1:4
        forward( turtle, 100 )
        turn( turtle, -90 )
    end
end
```

Loop (ciclo) che genera un quadrato

Ecco un'istruzione `for` che disegna un quadrato:

```
turtle = Turtle()
@svg begin
    for i in 1:4
        forward( turtle, 100 )
        turn( turtle, -90 )
    end
end
```



Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Un'istruzione `for` è anche chiamata loop perché il flusso di esecuzione scorre attraverso il corpo e poi torna all'inizio.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Un'istruzione `for` è anche chiamata loop perché il flusso di esecuzione scorre attraverso il corpo e poi torna all'inizio.

- In questo caso, esegue il corpo quattro volte.

Questa versione ha anche l'effetto di lasciare la tartaruga di nuovo nella posizione di partenza, rivolta nella direzione di partenza.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Un'istruzione `for` è anche chiamata loop perché il flusso di esecuzione scorre attraverso il corpo e poi torna all'inizio.

- In questo caso, esegue il corpo quattro volte.
- Questa versione è in realtà un po' diversa dal precedente codice di disegno del quadrato perché fa un altro giro dopo aver disegnato l'ultimo lato del quadrato.

Questa versione ha anche l'effetto di lasciare la tartaruga di nuovo nella posizione di partenza, rivolta nella direzione di partenza.

Sintassi di un'istruzione for

La sintassi di un'istruzione `for` è simile alla definizione di una funzione.

1. Ha un'intestazione e un corpo che termina con la parola chiave `end`.
2. Il corpo può contenere un numero qualsiasi di dichiarazioni o istruzioni.

Un'istruzione `for` è anche chiamata loop perché il flusso di esecuzione scorre attraverso il corpo e poi torna all'inizio.

- In questo caso, esegue il corpo quattro volte.
- Questa versione è in realtà un po' diversa dal precedente codice di disegno del quadrato perché fa un altro giro dopo aver disegnato l'ultimo lato del quadrato.
- Il turn extra richiede più tempo, ma semplifica il codice se facciamo la stessa cosa ogni volta nel ciclo.

Questa versione ha anche l'effetto di lasciare la tartaruga di nuovo nella posizione di partenza, rivolta nella direzione di partenza.

Section 3

Esercizi

Exercises I

Serie di esercizi con le tartarughe: sono pensati per essere divertenti, ma svolgono anche una funzione. Mentre ci lavori, pensa a qual è il punto

Le sezioni seguenti contengono soluzioni agli esercizi, quindi non guardare fino a quando non hai finito (o almeno provato).

Esercizio 4-2 Scrivi una funzione chiamata `square` che accetta un parametro chiamato `t`, che è una `turtle`. Dovrebbe usare la "tartaruga" per disegnare un quadrato.

Exercises I

Serie di esercizi con le tartarughe: sono pensati per essere divertenti, ma svolgono anche una funzione. Mentre ci lavori, pensa a qual è il punto

Le sezioni seguenti contengono soluzioni agli esercizi, quindi non guardare fino a quando non hai finito (o almeno provato).

Esercizio 4-2 Scrivi una funzione chiamata `square` che accetta un parametro chiamato `t`, che è una `turtle`. Dovrebbe usare la "tartaruga" per disegnare un quadrato.

Esercizio 4-3 Scrivi una chiamata di funzione che passi "t" come argomento a `square`, quindi esegui di nuovo la macro**.

Exercises I

Serie di esercizi con le tartarughe: sono pensati per essere divertenti, ma svolgono anche una funzione. Mentre ci lavori, pensa a qual è il punto

Le sezioni seguenti contengono soluzioni agli esercizi, quindi non guardare fino a quando non hai finito (o almeno provato).

Esercizio 4-2 Scrivi una funzione chiamata `square` che accetta un parametro chiamato `t`, che è una `turtle`. Dovrebbe usare la "tartaruga" per disegnare un quadrato.

Esercizio 4-3 Scrivi una chiamata di funzione che passi "t" come argomento a `square`, quindi esegui di nuovo la macro**.

Esercizio 4-4 Aggiungi un altro parametro, chiamato "len", a "square". Modificare il corpo in modo che la lunghezza dei lati sia "len", quindi modificare la chiamata alla funzione per fornire un secondo argomento. Eseguire di nuovo la macro. Provare con un intervallo di valori per "len".

Exercises I

Serie di esercizi con le tartarughe: sono pensati per essere divertenti, ma svolgono anche una funzione. Mentre ci lavori, pensa a qual è il punto

Le sezioni seguenti contengono soluzioni agli esercizi, quindi non guardare fino a quando non hai finito (o almeno provato).

Esercizio 4-2 Scrivi una funzione chiamata `square` che accetta un parametro chiamato `t`, che è una `turtle`. Dovrebbe usare la "tartaruga" per disegnare un quadrato.

Esercizio 4-3 Scrivi una chiamata di funzione che passi "t" come argomento a `square`, quindi esegui di nuovo la macro**.

Esercizio 4-4 Aggiungi un altro parametro, chiamato "len", a "square". Modificare il corpo in modo che la lunghezza dei lati sia "len", quindi modificare la chiamata alla funzione per fornire un secondo argomento. Eseguire di nuovo la macro. Provare con un intervallo di valori per "len".

Esercizio 4-5 Crea una copia di "`square`" e cambia il nome in "`polygon`". Aggiungi un altro parametro chiamato `n` e modifica il corpo in modo che disegni un poligono regolare di *n* lati.

Exercises II

Esercizio 4-6 Scrivi una funzione chiamata "`circle`" che prenda una "`Turtle`", "`t`" e un raggio, "`r`", come parametri e che disegni un cerchio approssimativo chiamando "`polygon`" con una lunghezza e un numero di lati appropriati.

Testa la tua funzione con un intervallo di valori di "`r`".

Exercises II

Esercizio 4-6 Scrivi una funzione chiamata "`circle`" che prenda una "`Turtle`", "`t`" e un raggio, "`r`", come parametri e che disegni un cerchio approssimativo chiamando "`polygon`" con una lunghezza e un numero di lati appropriati.

Testa la tua funzione con un intervallo di valori di "`r`".

Exercises II

Esercizio 4-6 Scrivi una funzione chiamata "`circle`" che prenda una "`Turtle`", "`t`" e un raggio, "`r`", come parametri e che disegni un cerchio approssimativo chiamando "`polygon`" con una lunghezza e un numero di lati appropriati.

Testa la tua funzione con un intervallo di valori di "`r`".

Esercizio 4-7 Crea una versione più generale di `circle` chiamata `arc` che accetta un parametro aggiuntivo `angle`, che determina quale frazione di cerchio disegnare.

`angle` è in unità di gradi, quindi quando `angle = 360`, `arc` dovrebbe disegnare un cerchio completo. (test !)

SUGGERIMENTI

```
len * n == circonferenza
```

Exercises II

Esercizio 4-6 Scrivi una funzione chiamata "`circle`" che prenda una "`Turtle`", "`t`" e un raggio, "`r`", come parametri e che disegni un cerchio approssimativo chiamando "`polygon`" con una lunghezza e un numero di lati appropriati.

Testa la tua funzione con un intervallo di valori di "`r`".

Esercizio 4-7 Crea una versione più generale di `circle` chiamata `arc` che accetta un parametro aggiuntivo `angle`, che determina quale frazione di cerchio disegnare.

`angle` è in unità di gradi, quindi quando `angle = 360`, `arc` dovrebbe disegnare un cerchio completo. (test !)

SUGGERIMENTI

- Calcola la circonferenza del cerchio e assicurati che

```
len * n == circonferenza
```

Exercises II

Esercizio 4-6 Scrivi una funzione chiamata "`circle`" che prenda una "`Turtle`", "`t`" e un raggio, "`r`", come parametri e che disegni un cerchio approssimativo chiamando "`polygon`" con una lunghezza e un numero di lati appropriati.

Testa la tua funzione con un intervallo di valori di "`r`".

Esercizio 4-7 Crea una versione più generale di `circle` chiamata `arc` che accetta un parametro aggiuntivo `angle`, che determina quale frazione di cerchio disegnare.

`angle` è in unità di gradi, quindi quando `angle = 360`, `arc` dovrebbe disegnare un cerchio completo. (test !)

SUGGERIMENTI

- Calcola la circonferenza del cerchio e assicurati che

```
len * n == circonferenza
```

Section 4

Incapsulamento

Incapsulamento 1/2

Il primo esercizio chiede di inserire il tuo codice di disegno del quadrato in una definizione di funzione e quindi chiamare la funzione, passando la `tartaruga` come parametro.

Ecco una soluzione:

```
function square(t)
    for i in 1:4
        forward(t, 100)
        turn(t, -90)
    end
end
turtle = Turtle()
@svg begin
    square(turtle)
end
```

Le istruzioni più interne, `forward` e `turn` sono rientrate due volte per mostrare che sono all'interno del ciclo `for`, che è all'interno della definizione di `function`.

Incapsulamento 2/2

All'interno della `function`, `t` si riferisce alla stessa `Turtle`, quindi `turn (t, -90)` ha lo stesso effetto di `*turn (turtle, -90)**`.

In tal caso, perché non chiamare il parametro `turtle`? L'idea è che `t` può essere qualsiasi `Turtle`, non solo `turtle`, quindi potremmo creare una seconda tartaruga e passarla come argomento a `square`:

```
turtle2 = Turtle()
@svg begin
    square(turtle)
end,
```

Racchiudere un pezzo di codice in una funzione è chiamato incapsulamento.

Uno dei vantaggi dell'incapsulamento è che attribuisce un nome al codice, che serve come una sorta di documentazione.
Un altro vantaggio è che se riutilizzi il codice, è più conciso chiamare una funzione due volte che copiare e incollare il corpo!

Section 5

Generalizzazione

Example 1/2

Il passo successivo è aggiungere un parametro "len" a "quadrato".

Ecco una soluzione:

```
function square(t, len)
    for i in 1:4
        forward(t, len)
        turn(t, -90)
    end
end
turtle = Turtle()
@svg begin
    square(turtle, 100)
end
```

L'aggiunta di un parametro a una funzione è chiamata generalizzazione perché rende la funzione più generale.

Nella versione precedente, il "quadrato" è sempre della stessa dimensione; in questa versione può essere di ogni misura

Example 2/2

Il passaggio successivo è ancora una generalizzazione.

Invece di disegnare quadrati, "polygon" disegna poligoni regolari con qualsiasi numero di lati. Ecco una soluzione:

```
function polygon(t, n, len)
    angle = 360 / n
    for i in 1:n
        forward(t, len)
        turn(t, -angle)
    end
end
turtle = Turtle()
@svg begin
    polygon(turtle, 7, 70)
end
```

Questo esempio disegna un poligono di 7 lati, con lato di lunghezza 70.

Section 6

Progettazione dell'interfaccia

Refactoring del codice

Il passo successivo è scrivere "circle", che prende un raggio, "r", come parametro. Ecco una semplice soluzione che usa `polygon` per disegnare un poligono da 50 lati:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = 50
    len = circumference / n
    polygon(t, n, len)
end
```

- La prima riga calcola la "circonferenza" di un cerchio con raggio "r" utilizzando la formula $2\pi r$.

Refactoring del codice

Il passo successivo è scrivere "circle", che prende un raggio, "r", come parametro. Ecco una semplice soluzione che usa `polygon` per disegnare un poligono da 50 lati:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = 50
    len = circumference / n
    polygon(t, n, len)
end
```

- La prima riga calcola la "circonferenza" di un cerchio con raggio "r" utilizzando la formula $2\pi r$.
- "n" è il numero di segmenti di linea nella nostra approssimazione di un cerchio, quindi "len" è la lunghezza di ogni segmento.

Refactoring del codice

Il passo successivo è scrivere "circle", che prende un raggio, "r", come parametro. Ecco una semplice soluzione che usa `polygon` per disegnare un poligono da 50 lati:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = 50
    len = circumference / n
    polygon(t, n, len)
end
```

- La prima riga calcola la "circonferenza" di un cerchio con raggio "r" utilizzando la formula $2\pi r$.
- "n" è il numero di segmenti di linea nella nostra approssimazione di un cerchio, quindi "len" è la lunghezza di ogni segmento.
- Pertanto, "poligono" disegna un poligono a 50 lati che approssima un cerchio con raggio "r".

Refactoring del codice

Il passo successivo è scrivere "circle", che prende un raggio, "r", come parametro. Ecco una semplice soluzione che usa `polygon` per disegnare un poligono da 50 lati:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = 50
    len = circumference / n
    polygon(t, n, len)
end
```

- La prima riga calcola la "circonferenza" di un cerchio con raggio "r" utilizzando la formula $2\pi r$.
- "n" è il numero di segmenti di linea nella nostra approssimazione di un cerchio, quindi "len" è la lunghezza di ogni segmento.
- Pertanto, "poligono" disegna un poligono a 50 lati che approssima un cerchio con raggio "r".

Refactoring del codice

Il passo successivo è scrivere "circle", che prende un raggio, "r", come parametro. Ecco una semplice soluzione che usa `polygon` per disegnare un poligono da 50 lati:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = 50
    len = circumference / n
    polygon(t, n, len)
end
```

- La prima riga calcola la "circonferenza" di un cerchio con raggio "r" utilizzando la formula $2\pi r$.
- "n" è il numero di segmenti di linea nella nostra approssimazione di un cerchio, quindi "len" è la lunghezza di ogni segmento.
- Pertanto, "poligono" disegna un poligono a 50 lati che approssima un cerchio con raggio "r".

Una limitazione di questa soluzione è che "n" è una costante, il che significa che per cerchi molto grandi i segmenti di linea sono troppo lunghi e per cerchi piccoli perdiamo tempo a disegnare segmenti molto piccoli.

Una soluzione sarebbe generalizzare la funzione prendendo "n" come parametro. Questo darebbe all'utente (chiunque chiami `circle`) più controllo, ma l'interfaccia sarebbe meno pulita.

Making better

L'interfaccia di una funzione è un riassunto di come viene utilizzata: quali sono i parametri? Cosa fa la funzione? E qual è il valore di ritorno? Un'interfaccia è "pulita" se consente al chiamante di fare ciò che desidera senza occuparsi di dettagli non necessari.

- In questo esempio, "r" appartiene all'interfaccia perché specifica il cerchio da disegnare.

Making better

L'interfaccia di una funzione è un riassunto di come viene utilizzata: quali sono i parametri? Cosa fa la funzione? E qual è il valore di ritorno? Un'interfaccia è "pulita" se consente al chiamante di fare ciò che desidera senza occuparsi di dettagli non necessari.

- In questo esempio, "r" appartiene all'interfaccia perché specifica il cerchio da disegnare.
- *"n" è meno appropriato perché riguarda i dettagli di come il cerchio dovrebbe essere reso.

Making better

L'interfaccia di una funzione è un riassunto di come viene utilizzata: quali sono i parametri? Cosa fa la funzione? E qual è il valore di ritorno? Un'interfaccia è "pulita" se consente al chiamante di fare ciò che desidera senza occuparsi di dettagli non necessari.

- In questo esempio, "r" appartiene all'interfaccia perché specifica il cerchio da disegnare.
- *"n" è meno appropriato perché riguarda i dettagli di come il cerchio dovrebbe essere reso.

Making better

L'interfaccia di una funzione è un riassunto di come viene utilizzata: quali sono i parametri? Cosa fa la funzione? E qual è il valore di ritorno? Un'interfaccia è "pulita" se consente al chiamante di fare ciò che desidera senza occuparsi di dettagli non necessari.

- In questo esempio, "r" appartiene all'interfaccia perché specifica il cerchio da disegnare.
- *"n" è meno appropriato perché riguarda i dettagli di come il cerchio dovrebbe essere reso.

Piuttosto che ingombrare l'interfaccia, è meglio scegliere un valore appropriato di "n" a seconda della circonferenza:

```
function circle(t, r)
    circumference = 2 * pi * r
    n = trunc(circumference / 3) + 3
    len = circumference / n
    polygon(t, n, len)
end
```

Ora il numero di segmenti è un numero intero vicino a "circumference / 3", quindi la lunghezza di ogni segmento è circa 3, che è abbastanza piccola da far sembrare i cerchi belli, ma abbastanza grandi da essere efficienti e accettabili per cerchi di qualsiasi dimensione.

L'aggiunta di 3 a "n" garantisce che il poligono abbia** almeno 3 lati**.