Teaching Functional Programming at ELTE

ICTPL 2024

István Bozó

Eötvös Loránd University, Hungary

April 5, 2024

Courses taught

- Teaching functional programming since 2009
- Functional programing - BSc, Haskell
- Programming languages 2. - teacher education, Haskell
- Functional languages - MSc, Haskell
- Design of Distributed Systems - MSc, Erlang
- *Programming with functions - students from other faculties, Haskell*

Difficulties in teaching FP

- Learning programming "from scratch".
- Requires a different way of thinking.
- First "unlearning" things to learn new things.
- Understanding: polymorphism, currying, higher-order functions, etc.

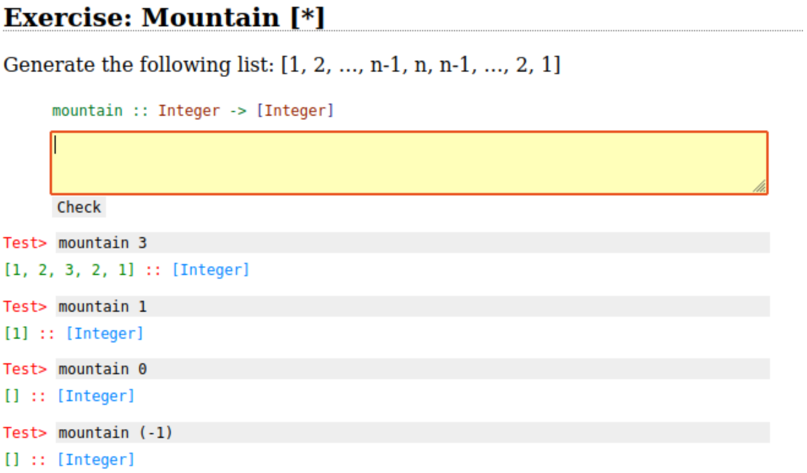
Syllabus of Haskell courses

- Simple expressions, operators, function applications
- Operator precedence and associativity
- Function definitions, pattern matching, guards (branching)
- Recursion
- Ad-hoc (type classes) and parametric polymorphism
- Higher-order functions
- User-defined types and synonyms

Used tools

- A text editor or IDE of choice/preference (Geany, NotePad++, VS code, etc.)
- GHCi - Haskell interpreter
- Online documentation (Hoogle)
- *ActiveHS*

ActiveHs - playing with Haskell



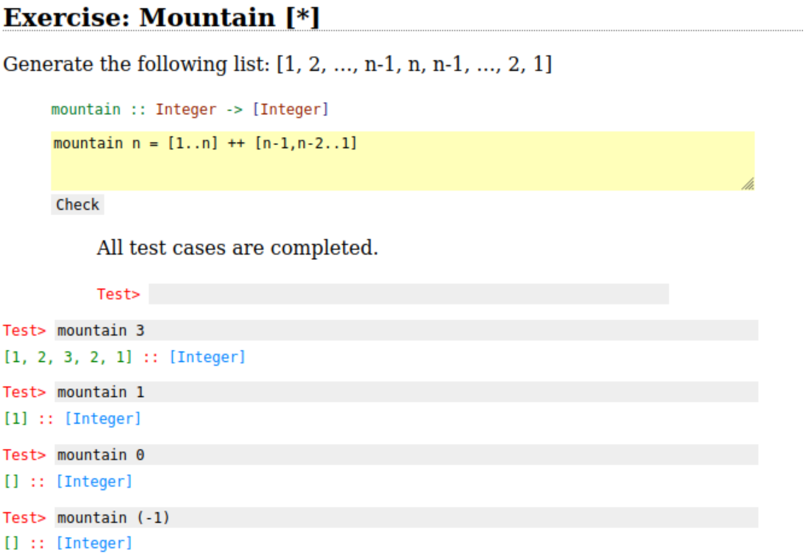
Exercise: Mountain [*]

Generate the following list: [1, 2, ..., n-1, n, n-1, ..., 2, 1]

```
mountain :: Integer -> [Integer]
```

```
Test> mountain 3
[1, 2, 3, 2, 1] :: [Integer]

Test> mountain 1
[1] :: [Integer]

Test> mountain 0
[] :: [Integer]

Test> mountain (-1)
[] :: [Integer]
```

ActiveHs - playing with Haskell



Exercise: Mountain [*]

Generate the following list: [1, 2, ..., n-1, n, n-1, ..., 2, 1]

```
mountain :: Integer -> [Integer]
```

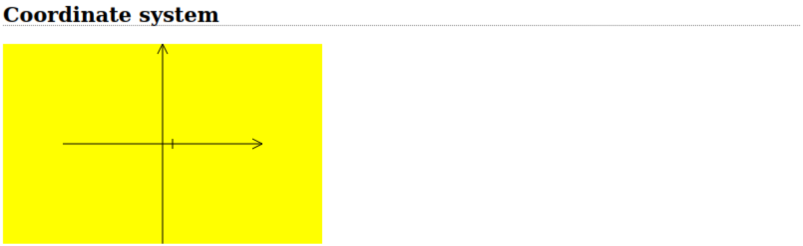
```
mountain n = [1..n] ++ [n-1,n-2..1]
```

Check

All test cases are completed.

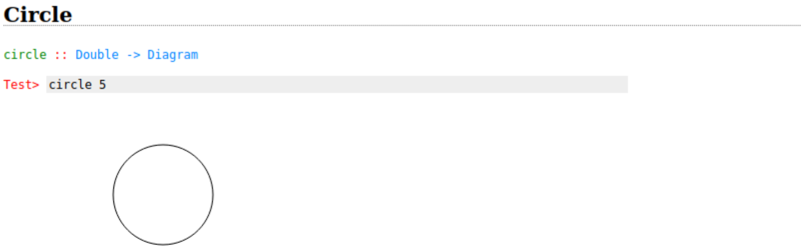
```
Test>

Test> mountain 3
[1, 2, 3, 2, 1] :: [Integer]

Test> mountain 1
[1] :: [Integer]

Test> mountain 0
[] :: [Integer]

Test> mountain (-1)
[] :: [Integer]
```

ActiveHs - playing with Haskell



Coordinate system

ActiveHs - playing with Haskell



Circle

```
circle :: Double -> Diagram
```

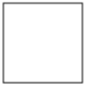
```
Test> circle 5
```

ActiveHs - playing with Haskell

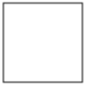
Rectangle

```
rect :: Double -> Double -> Diagram
```

```
Test> rect 8 8
```



```
Test> rect (4+4) (5+3)
```

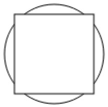


ActiveHs - playing with Haskell

Joining two diagrams

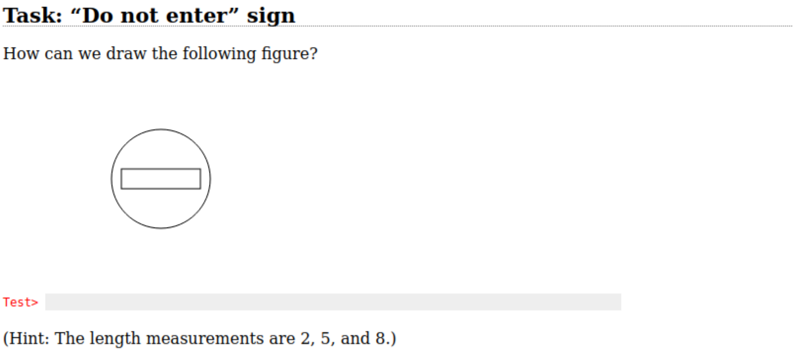
The <|> operator combines two diagrams into one diagram.

```
(<|>) :: Diagram -> Diagram -> Diagram

Test> circle 5 <|> rect 8 8
```



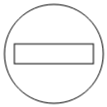
Notice that the square is on top of the circle. How can we put the circle on top of the square?

ActiveHs - playing with Haskell



Task: “Do not enter” sign

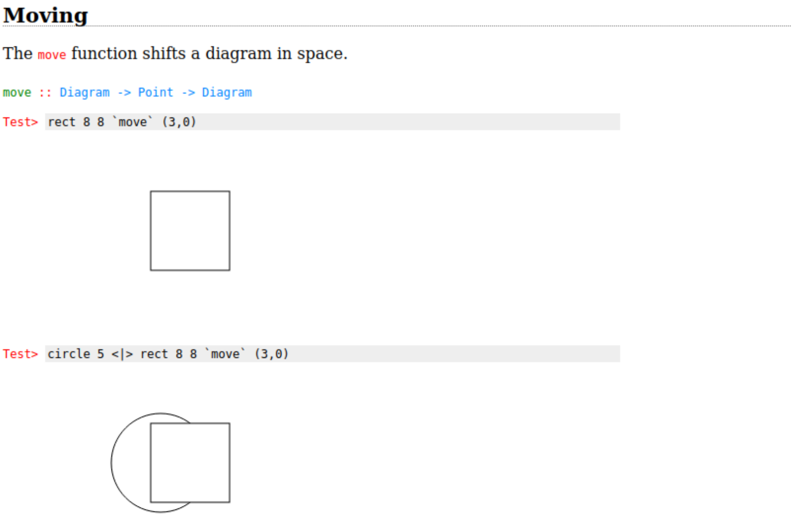
How can we draw the following figure?



```
Test>
```

(Hint: The length measurements are 2, 5, and 8.)

ActiveHs - playing with Haskell



Moving

The `move` function shifts a diagram in space.

```
move :: Diagram -> Point -> Diagram
```

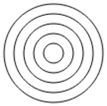
```
Test> rect 8 8 `move` (3,0)
```

```
Test> circle 5 <|> rect 8 8 `move` (3,0)
```

ActiveHs - playing with Haskell

Combining multiple diagrams

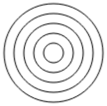
```
Test> circle 5 <|> circle 4 <|> circle 3 <|> circle 2 <|> circle 1
```



Instead of using `<|>`, we can use `union` to combine diagrams. `union` takes a list of diagrams.

```
union :: [Diagram] -> Diagram
```

```
Test> union [circle 5, circle 4, circle 3, circle 2, circle 1]
```

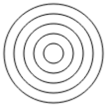


ActiveHs - playing with Haskell

Using list comprehensions

We can use list comprehensions to express diagrams.

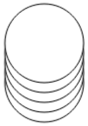
```
Test> union [circle i | i<-[5,4..1]]
```



ActiveHs - playing with Haskell

Task: stack of coins

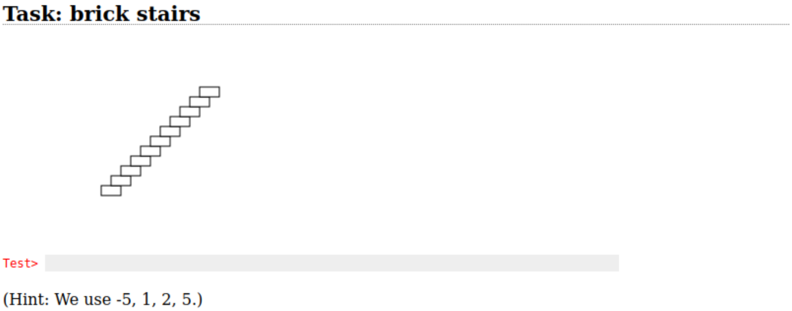
How can we draw this stack of coins using list comprehensions?



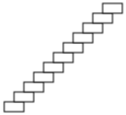
```
Test>
```

(Hint: The circles’ diameter is 4.)

ActiveHs - playing with Haskell



Task: brick stairs

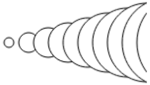


Test>

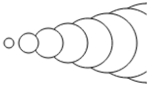
(Hint: We use -5, 1, 2, 5.)

ActiveHs - playing with Haskell

Task: Circles



Test>

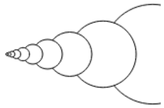


Test>

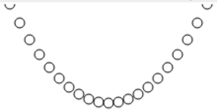
ActiveHs - playing with Haskell

Other interesting diagrams

```
Test> union [circle (1.5**x) `move` (3*1.5**x, 0) | x <- [5,4.. -5]]
```

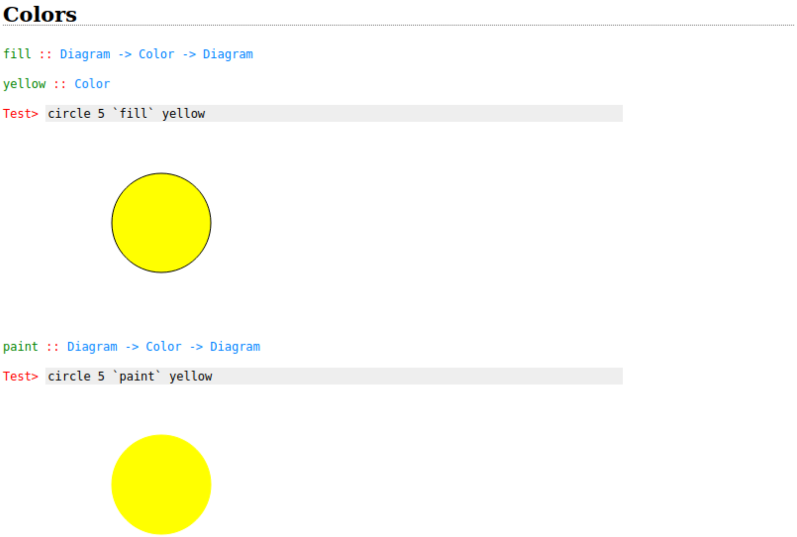


```
Test> union [circle 0.5 `move` (x, x^2 / 10) | x<-[-20..20]]
```



```
Test> union [circle (x/5) `move` (x*sin x, x*cos x) | x<-[0,pi/6..4*pi]]
```

ActiveHs - playing with Haskell

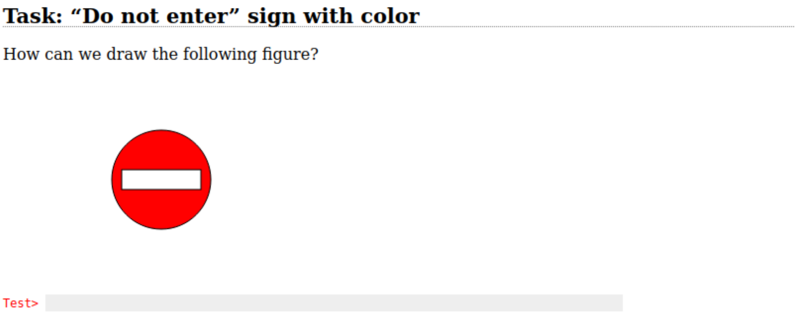


Colors

```
fill :: Diagram -> Color -> Diagram

yellow :: Color

Test> circle 5 `fill` yellow
```

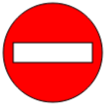
```
paint :: Diagram -> Color -> Diagram

Test> circle 5 `paint` yellow
```

ActiveHs - playing with Haskell



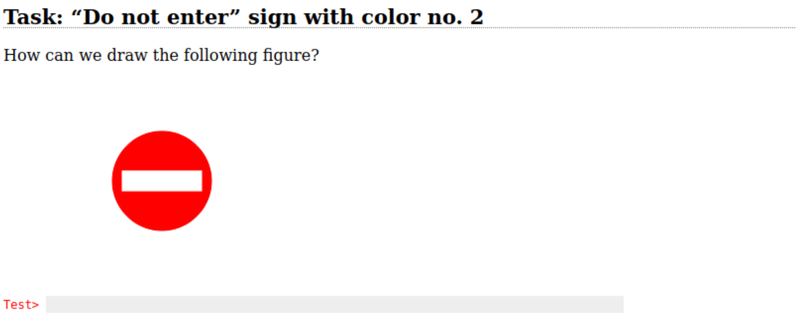
Task: "Do not enter" sign with color

How can we draw the following figure?



```
Test>
```

ActiveHs - playing with Haskell

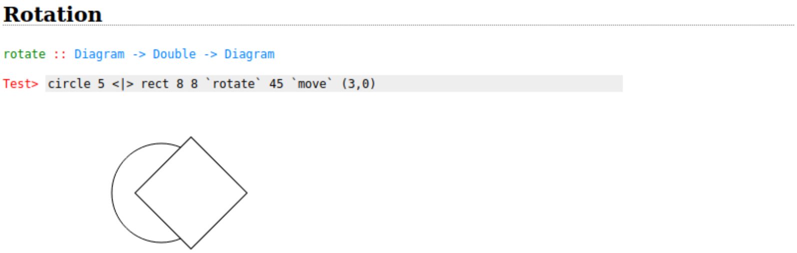


Task: “Do not enter” sign with color no. 2

How can we draw the following figure?

Test>

ActiveHs - playing with Haskell

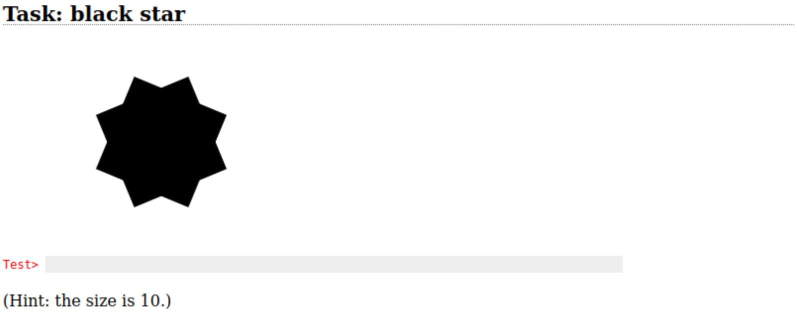


Rotation

```
rotate :: Diagram -> Double -> Diagram
```

```
Test> circle 5 <|> rect 8 8 `rotate` 45 `move` (3,0)
```

ActiveHs - playing with Haskell

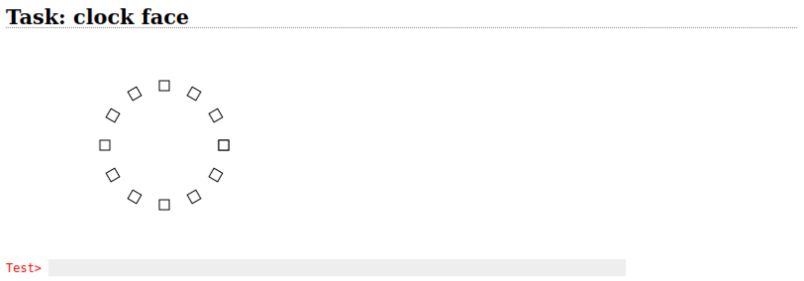


Task: black star

Test>

(Hint: the size is 10.)

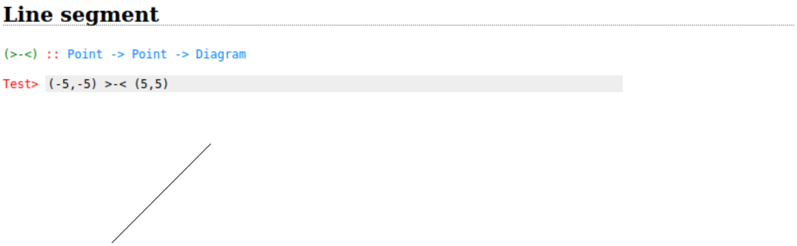
ActiveHs - playing with Haskell



Task: clock face

```
Test>
```

ActiveHs - playing with Haskell

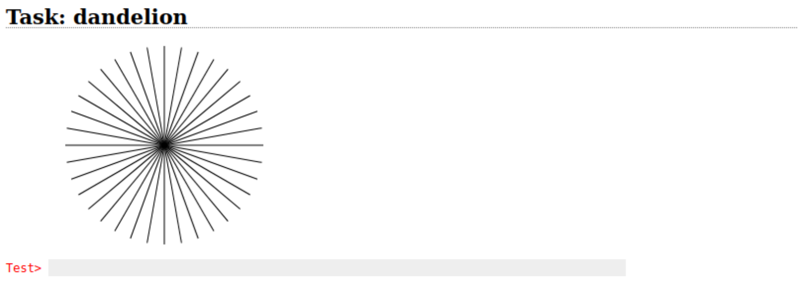


Line segment

```
(>-<) :: Point -> Point -> Diagram
```

```
Test> (-5,-5) >-< (5,5)
```

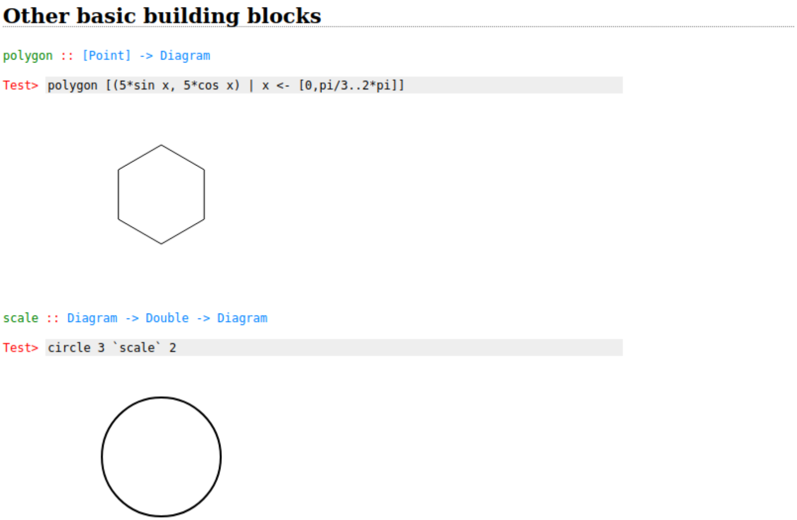
ActiveHs - playing with Haskell



Task: dandelion

```
Test>
```

ActiveHs - playing with Haskell

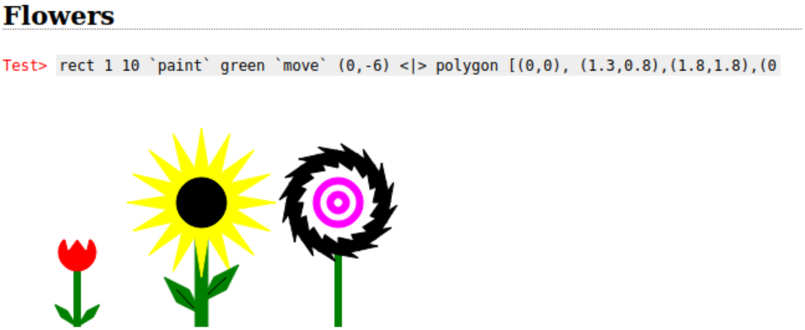


Other basic building blocks

```
polygon :: [Point] -> Diagram

Test> polygon [(5*sin x, 5*cos x) | x <- [0,pi/3..2*pi]]
```

```
scale :: Diagram -> Double -> Diagram

Test> circle 3 `scale` 2
```

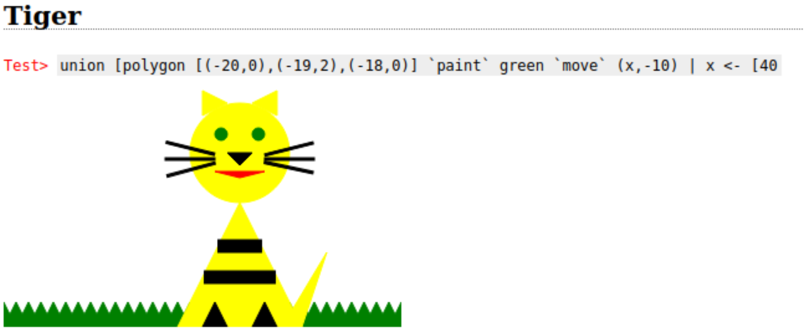
"There's a Child in All of Us!"



Flowers

```
Test> rect 1 10 `paint` green `move` (0,-6) <|> polygon [(0,0), (1.3,0.8),(1.8,1.8),(0
```

"There's a Child in All of Us!"



Tiger

```
Test> union [polygon [(-20,0),(-19,2),(-18,0)] `paint` green `move` (x,-10) | x <- [40
```

"There's a Child in All of Us!"

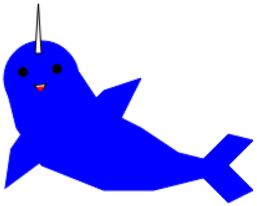
Unicorn

```
Test> let formula r n = union [circle r `move` (x,0) |  x <- [0, r .. (n-1)*r]] in rec
```

"There's a Child in All of Us!"

Narwhal

```
Test> (circle 3 `paint` blue `move` (-5,5) <|> polygon [(-8,5), (-7,1), (-5,-2), (0,-4
```

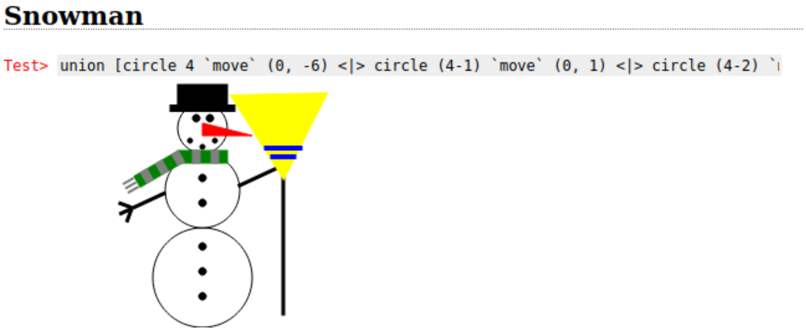


"There's a Child in All of Us!"

Cat

```
Test> circle 8 `paint` gray <|> circle 1 `paint` black `move` (-2.5,2) <|> circle 1 `p
```

"There's a Child in All of Us!"



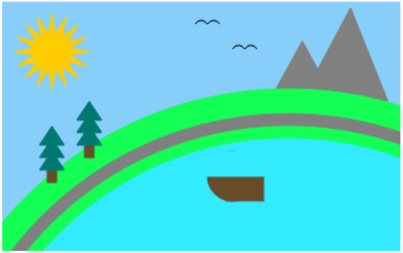
Snowman

```
Test> union [circle 4 `move` (0, -6) <|> circle (4-1) `move` (0, 1) <|> circle (4-2) `
```

"There's a Child in All of Us!"

Landscape

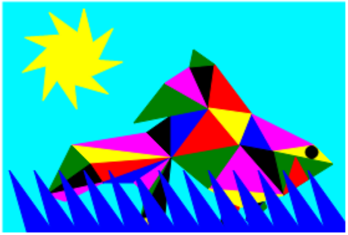
```
Test> union [rect 32 20 `paint` (rgb 0.53 0.81 0.98), polygon [(5.5,3), (10.5,3), (7.5
```



"There's a Child in All of Us!"

Rainbow fish

```
Test> (rect 35 25 `paint` (rgb 0 0.969 1) `move`(4,0)<|>  polygon [(0,0), (3,4), (5.5,
```



"There's a Child in All of Us!"

Panda

```
Test> union [circle 1.5 `move` (-2.9, 5.25) `paint` black, circle 1.5 `move` (2.9, 5.2
```

"There's a Child in All of Us!"

Dragon

```
Test> union [polygon [(-1,-1), (-3,-5), (-3.5,-7), (-3,-8.5), (-2,-9), (0.5,-9), (2,-9
```