1B: Elements of Programming

CS1101S: Programming Methodology

Martin Henz

August 12, 2016

Elements of Programming

- Primitives
- Combination
- Abstraction

Elements of Programming: Primitives

- Primitives
  - Primitive number expressions
  - Primitive boolean expressions (`true`, `false`)
  - New: *String* expressions

Elements of Programming

- Combination
  - Arithmetic operators
  - Comparison operators
  - New: *Boolean operators*

Elements of Programming

- Abstraction
  - Names
  - Functions

A function can be a “black box”

Example

```
Math.floor
```

- Input: any number
- Output: the largest integer that is not larger than the number

How does it work?

Do we need to know in order to use it?

Primitives

Some simple pictures such as

- `rcross_bb`
- `sail_bb`
- `corner_bb`
- `nova_bb`
- `heart_bb`

Abstraction

Transformations

Functions that take a picture and produce a new picture from it

Example transformation

Turn a given picture a quarter turn right:
`quarter_turn_right`

Example in JediScript

```
quarter_turn_right(quarter_turn_right(sail_bb))
```

Abstraction: Functions and naming

```
function turn_upside_down(picture) {
    return quarter_turn_right(
                quarter_turn_right(picture));
}
```

Abstraction: Functions and naming

```
function quarter_turn_left(picture) {
    return turn_upside_down(
                quarter_turn_right(picture));
}
```

Combination: stacking

```
stack(rcross_bb, sail_bb);
```

Your turn: Combination and abstraction

```
function beside(picture) {
    ???
}
```

Your turn: Combination and abstraction

```
function beside(picture1, picture2) {
    return quarter_turn_right(
             stack(quarter_turn_left(picture2),
                   quarter_turn_left(picture1)));
}
```

Naming: a more complex picture

```
var my_cross =
    stack(beside(quarter_turn_right(rcross_bb),
                 turn_upside_down(rcross_bb)),
          beside(rcross_bb,
                 quarter_turn_left(rcross_bb)));
```

Abstraction: making a cross from anything

```
function make_cross(picture) {
    return stack(beside(
                        quarter_turn_right(picture),
                        turn_upside_down(picture)),
                    beside(
                        picture,
                        quarter_turn_left(picture)));
}
```

Combination: repeating the pattern

```
make_cross( make_cross( nova ) )
```

Abstraction: repeating the pattern *n* times

```
repeat_pattern(4, make_cross, rcross_bb)
```

Surprising fact

On Wednesday, we will define this abstraction in The Source.

Abstraction: stacking *n* times

```
stackn(5, heart_bb)
```

Defining `stack_n`

Like `repeat_pattern`, we will define `stackn` in The Source on Wednesday.

Combination: rectangular quilting

```
stackn(5,
       quarter_turn_right(
           stackn(5,quarter_turn_left(nova_bb))))
```

Abstraction: rectangular quilting

```
function quilt(n, m, picture) {
    return stackn(n,
                  quarter_turn_right(
                     stackn(m,
                            quarter_turn_left(
                               picture))));
}
```

Reflection

- No idea how the primitives work
  Example: `heart_bb`
- No idea how the given simple combinations work
  Example: `quarter_turn_right`
- Yet we can generate complex pictures, through combination and abstraction