B1: Runology

CS1101S: Programming Methodology

Martin Henz

August 13, 2021

First Mission: Rune Trials

First Mission: Rune Trials

- ...will be released at 10:45 **today**.

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP
- **Good news:** Path P1A already carries XP (we make sure everyone gets 100)

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP
- **Good news:** Path P1A already carries XP (we make sure everyone gets 100)
- Path P1B will be released at 10:45 today.

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP
- **Good news:** Path P1A already carries XP (we make sure everyone gets 100)
- Path P1B will be released at 10:45 today.
- You *can* discuss paths, missions, and quests with your friends and avengers.

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP
- **Good news:** Path P1A already carries XP (we make sure everyone gets 100)
- Path P1B will be released at 10:45 today.
- You *can* discuss paths, missions, and quests with your friends and avengers.
- You *must* do the actual programming yourself. Do *not* copy anyone's programs, from other students or from previous batches.

First Mission: Rune Trials

- ...will be released at 10:45 **today**.
- Missions are submitted individually and carry significant XP.
- Paths also carry XP
- **Good news:** Path P1A already carries XP (we make sure everyone gets 100)
- Path P1B will be released at 10:45 today.
- You *can* discuss paths, missions, and quests with your friends and avengers.
- You *must* do the actual programming yourself. Do *not* copy anyone's programs, from other students or from previous batches.
- Submitting other people's work as your own is a serious offence and will be severely punished by NUS.

Assessment in CS1101S

- 18%: Missions, Quests, Paths and other activities (XP)
- 5%: Studio attendance and effort
- 5%: Reflection attendance
- 6%: Reading Assessment 1, Week 4
- 3%: Mastery Check 1, any time
- 12%: Midterm Assessment, Week 7
- 6%: Reading Assessment 2, Week 10
- 12%: Practical Assessment, Week 13
- 3%: Mastery Check 2, any time
- 30%: Final Assessment

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

Early submission bonus

Submit Missions and Quests within 3 days of release: 75 XP bonus.

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

Early submission bonus

Submit Missions and Quests within 3 days of release: 75 XP bonus. Submit Paths within 1 day of release: 25 XP bonus.

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

Early submission bonus

Submit Missions and Quests within 3 days of release: 75 XP bonus. Submit Paths within 1 day of release: 25 XP bonus.

Contests

Look out for announcements. Winners & runners-up receive XP.

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

Early submission bonus

Submit Missions and Quests within 3 days of release: 75 XP bonus. Submit Paths within 1 day of release: 25 XP bonus.

Contests

Look out for announcements. Winners & runners-up receive XP. You will also earn XP for participating in the contest voting.

18%: based on XP in Source Academy

Your homework

Homework (Paths, Missions, Quests) earn XP.

Early submission bonus

Submit Missions and Quests within 3 days of release: 75 XP bonus. Submit Paths within 1 day of release: 25 XP bonus.

Contests

Look out for announcements. Winners & runners-up receive XP. You will also earn XP for participating in the contest voting.

Miscellaneous special topics

Extra XP to deserving individuals; look out for announcements.

CS1010R

CS1101S as 4 MC

18% XP component: you get full marks if you reach our expected threshold of 36000 XP. Less XP means less marks.

CS1010R

CS1101S as 4 MC

18% XP component: you get full marks if you reach our expected threshold of 36000 XP. Less XP means less marks.

XP beyond threshold...

...count for CS1010R: the more XP, the higher the grade for CS1010R

CS1010R

CS1101S as 4 MC

18% XP component: you get full marks if you reach our expected threshold of 36000 XP. Less XP means less marks.

XP beyond threshold...

...count for CS1010R: the more XP, the higher the grade for CS1010R

CS1010R has 1 MC

If you meet the bar, you will be pre-allocated CS1010R in Sem 2, and can drop it if you are not happy with the grade.

Racap: Elements of Programming (1.1)

Racap: Elements of Programming (1.1)

- Primitives: things like -42

Racap: Elements of Programming (1.1)

- Primitives: things like `-42`
- Combination: things like `-42 * 7`

Racap: Elements of Programming (1.1)

- Primitives: things like `-42`
- Combination: things like `-42 * 7`
- Name abstraction: things like `const size = 2;`

Racap: Elements of Programming (1.1)

- Primitives: things like `-42`
- Combination: things like `-42 * 7`
- Name abstraction: things like `const size = 2;`
- Functional abstraction: more on this today

A function can be a "black box"

Example

`math_sqrt`

A function can be a "black box"

Example

`math_sqrt`

- Input: any number
- Output: a number whose square is the input number

A function can be a "black box"

Example

`math_sqrt`

- Input: any number
- Output: a number whose square is the input number

Function application

```
math_sqrt(15);
```

Means of Abstraction: Compound functions

Example

```
function square(x) {
    return x * x;
}
```

Means of Abstraction: Compound functions

Example

```
function square(x) {
    return x * x;
}
```

What does this statement mean?

Like constant declarations, this *function declaration* declares a name, here `square`. The value associated with the name is a function that takes an argument `x` and produces (returns) the result of multiplying `x` by itself.

Primitives

Some simple runes such as

- `rcross`
- `sail`
- `corner`
- `nova`
- `heart`
- `show` is a primitive operation that displays a rune in the REPL.

Primitive Operations

Transformations

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

Example transformation

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

Example in Source

```
quarter_turn_right(quarter_turn_right(sail))
```

Abstraction: Functions and naming

```
function turn_upside_down(rune) {
    return quarter_turn_right(
                quarter_turn_right(rune));
}
// example use:
show(turn_upside_down(heart));
```

Abstraction: Functions and naming (continued)

```
function quarter_turn_left(rune) {
    return quarter_turn_right(
                turn_upsidedown(rune));
}
// example use:
show(quarter_turn_left(heart));
```

Combination operation: stacking

```
stack(rcross, sail);
```

Your turn: Combination abstraction

```
function beside(rune1, rune2) {
    ???
}

// example use:
show(beside(rcross, sail));
// should show rcross on the left
//             and sail on the right
```

Solution: Combination abstraction

```
function beside(rune1, rune2) {
    return quarter_turn_left(
             stack(quarter_turn_right(rune1),
                   quarter_turn_right(rune2)));
}
```

Combination: stacking something *n* times

```
stackn(5, heart);
```

Lecture L2

We will define `stackn` in Source, using rune primitives.

Naming and more combination: rectangular quilting

```
const my_quilt =
  stackn(5,
         quarter_turn_right(
           stackn(7,quarter_turn_left(heart))));
```

Abstraction: rectangular quilting

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

Combination: a more complex rune

```
stack(beside(quarter_turn_right(rcross),
             turn_upside_down(rcross)),
      beside(rcross,
             quarter_turn_left(rcross)));
```

Naming interesting things

```
const my_cross =
    stack(beside(quarter_turn_right(rcross),
                 turn_upside_down(rcross)),
          beside(rcross,
                 quarter_turn_left(rcross)));
```

Abstraction: making a cross from any rune

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

Combination: repeating the pattern

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

Abstraction: repeating the pattern *n* times

```
repeat_pattern(5, make_cross, rcross);
```

Lecture L2

Like `stackn`, we will define this abstraction in Source next Wednesday!

Recap

- We started with primitive constants.

Recap

- We started with primitive constants.
- No idea how they are rendered!

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!
  Example: `quarter_turn_right`

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!
  Example: `quarter_turn_right`
- Yet we can use primitives and combinations to generate complex runes.

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!
  Example: `quarter_turn_right`
- Yet we can use primitives and combinations to generate complex runes.
- **Abstractions** allow us to master complexity:

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!
  Example: `quarter_turn_right`
- Yet we can use primitives and combinations to generate complex runes.
- **Abstractions** allow us to master complexity:
  Naming

Recap

- We started with primitive constants.
- No idea how they are rendered!
  Example: `heart`
- We introduced primitive combinations.
- No idea how the primitive combinations work!
  Example: `quarter_turn_right`
- Yet we can use primitives and combinations to generate complex runes.
- **Abstractions** allow us to master complexity:
  Naming and **functional abstraction**