Structs, Vectors, and Classes in DSSL2

CS 214, Fall 2019
Welcome to DSSL2

- A close relative of Python
- But with data structures taken out!
  - (Otherwise, where’s the fun?)
- And with data structure building blocks added in
- Built on top of Racket
  - But quite different from Racket/BSL/ISL/…
Welcome to DSSL2

• Code organized in statements, functions, and classes
  ▶ Similar to C++

• Variables and data are mutable (= assignment)
  ▶ Similar to C++

• No explicit pointers (arrows) or memory management
  ▶ Similar to the 111 teaching languages

• No explicit types
  ▶ Similar to the 111 teaching languages

• (These also apply to Python)
DSSL2 expressions

$3 + 5$

# comments start with `#’ and continue
# to the end of the line

$6 \times (3 + 5)$

$1 + \text{len('hello')}$
DSSL2 expressions

3 + 5  # comments start with '#' and continue
# to the end of the line

6 * (3 + 5)

1 + 'hello'.len()
DSSL2 statements

```dssl2
let x = 5  # variable definitions use ‘let’ —
# this minor difference from Python
# helps avoid ambiguity and thus bugs
```

```dssl2
println(8 * x)
```

```dssl2
if condition:
# indentation matters! just like Python
do_some_stuff()
else:
do_other_stuff(x, y, z)
```
DSSL2 statements

```plaintext
let x = 5  # variable definitions use ‘let’ –
          # this minor difference from Python
          # helps avoid ambiguity and thus bugs

println(8 * x)  # an expression can also be a statement
```

DSSL2 statements

let x = 5  # variable definitions use ‘let’ –
# this minor difference from Python
# helps avoid ambiguity and thus bugs

println(8 * x)  # an expression can also be a statement

if condition:  # indentation matters! just like Python
do_some_stuff()
else:
do_other_stuff(x, y, z)
DSSL2 functions

# hypotenuse: Number Number -> Number
# Finds the length of the hypotenuse.
def hypotenuse(a, b):
    (a * a + b * b).sqrt()
DSSL2 functions

# hypotenuse: Number Number -> Number  
# Finds the length of the hypotenuse.
def hypotenuse(a, b):
    (a * a + b * b).sqrt()

# fact: Natural -> Natural  
# Computes the factorial of `n`.
def fact(n):
    if n == 0:
        1
    else:
        n * fact(n - 1)
DSSL2 functions

# hypotenuse: Number Number -> Number
# Finds the length of the hypotenuse.
def hypotenuse(a, b):
    (a * a + b * b).sqrt()

# fact: Natural -> Natural
# Computes the factorial of `n`.
def fact(n):
    if n == 0:
        1
    else:
        n * fact(n - 1)
DSSL2 assertions and test cases

An assertion errors (and stops your program) if it fails:

```python
# fails if fact(5) != 120:
assert_eq fact(5), 120
```

To run multiple tests, put your assertions in test blocks. When an error happens in a test block, it counts it as a failed test and continues running the program after the test block:

```python
test 'fact works':
    assert_eq fact(3), 6
    assert_eq fact(5), 120
```
DSSL2 assertions and test cases

An assertion errors (and stops your program) if it fails:

```
# fails if fact(5) != 120:
assert_eq fact(5), 120
```

To run multiple tests, put your assertions in test blocks. When an error happens in a test block, it counts it as a failed test and continues running the program after the test block:

```
test 'fact works':
    assert_eq fact(3), 6
    assert_eq fact(5), 120
```
DSSL2 programs

Every DSSL2 program starts with a #lang line, followed by any number of statements:

#lang dssl2

let CM_PER_INCH = 2.54

# Converts centimeters to inches.
def cm_to_inch(cm):
    cm / CM_PER_INCH

# Converts inches to centimeters.
def inch_to_cm(inches):
    inches * CM_PER_INCH

test 'round trip':
    assert_eq inch_to_cm(cm_to_inch(17)), 17
    assert_eq cm_to_inch(inch_to_cm(17)), 17
Vectors

• One of the key building blocks of data structures:

```
0 1 2 3 4 5 6 7 8 9
0 1 1 2 4 7 13 24 44 82
```

• Literal vector notation:

\[ [0, 1, 1, 2, 4, 7, 13, 24, 44, 82] \]
Vector operations

let v = [ 0, 1, 1, 2, 4, 7, 13, 24, 44, 82 ]
Vector operations

let v = [ 0, 1, 1, 2, 4, 7, 13, 24, 44, 82 ]

# you can give names to test cases
# and get nicer error messages than bare assumptions

test 'vector basics':
  assert_eq v[3], 2
  assert_eq v[6], 13
Vector operations

let v = [ 0, 1, 1, 2, 4, 7, 13, 24, 44, 82 ]

# you can give names to test cases
# and get nicer error messages than bare assumptions

test 'vector basics':
    assert_eq v[3], 2
    assert_eq v[6], 13

test 'vector set':
    v[6] = 23
    assert_eq v[6], 23
What if I want a really big vector?

- *Vector comprehensions* allow you to create a vector using a “description” rather than literal elements
  
  `[ 0; 1000000 ]
  
  - Creates a vector with 1000000 elements, all 0s
    - Much nicer than typing the whole thing!
  
  - Supports more complicated descriptions too, see the docs
Example: average

# average: Vector<Number> -> Number
# Averages the elements of a non-empty vector.
def average(vec):
    sum(vec) / vec.len()

# sum: Vector<Number> -> Number
# Sums the elements of a non-empty vector.
def sum(vec):
    let result = 0
    # for-each loop, like in C++
    # `v` becomes each element of the vector, in turn
    for v in vec:
        result = result + v
    return result
Discuss with your Neighbor

- Discuss what you already know about DSSL2
- And what is still mysterious
- In 2 minutes, let’s hear your questions
Structs

- Another key building block

```
struct posn:
  let x
  let y

# different ways to construct
posn { x: 12, y: -5 }
posn { x: 0, y: 0 }
posn(3, 4)
```
struct posn:
    let x
    let y

let p = posn(3, 4)
assert posn?(p)  # asserts that the result is true
assert_eq p.x, 3
assert_eq p.y, 4  # uses `.` notation, like C++

p.x = 6
assert_eq p.x, 6
assert_eq p.y, 4
struct employee:
    let id; let name; let position

let employees = [ employee( 928, "Alice", 4),
    employee(1089, "Bob", 6),
    employee( 14, "Carol", 6),
    employee( 546, "Dave", 6) ]
Working with structs and vectors

struct employee:
    let id; let name; let position

let employees = [
    employee(928, "Alice", 4),
    employee(1089, "Bob", 6),
    employee(14, "Carol", 6),
    employee(546, "Dave", 6),
]

QUIZ. Suppose we want to find out Carol’s position:
Working with structs and vectors

```swift
struct employee {
    let id; let name; let position
}

let employees = [
    employee(928, "Alice", 4),
    employee(1089, "Bob", 6),
    employee(14, "Carol", 6),
    employee(546, "Dave", 6),
]

QUIZ. Suppose we want to find out Carol’s position:

employees[2].position

QUIZ: How can we give her a promotion (from 6 to 5)?
struct employee:
    let id; let name; let position

let employees = [
    employee(928, "Alice", 4),
    employee(1089, "Bob", 6),
    employee(14, "Carol", 6),
    employee(546, "Dave", 6),
]

QUIZ. Suppose we want to find out Carol’s position:

employees[2].position

QUIZ: How can we give her a promotion (from 6 to 5)?

employees[2].position = 5
Generalizing

# promote_employee : Vector<Employee> Natural ->
# Decrements the position of the `index`th employee.
def promote_employee(employees, index):
    let emp = employees[index]
    # `emp` is not a copy! so we modify the original
    emp.position = emp.position - 1
Classes

- Structs and vectors are enough to represent any data.
- But data structures = representation + operations.
  - Classes allow us to combine the two.
- Classes \(\approx\) structs with methods.
  - A code organization mechanism to group data and operations together.
Our first class example

class Posn:
    
    let x
    # fields: initialized by
    let y
    # the constructor

    def __init__(self, x, y):
        # constructor: method
        self.x = x
        # with a special name
        self.y = y

    def get_x(self):
        # fields are private
        # `return` is optional
        return self.x

    def get_y(self):
        # `self` = receiver
        return self.y

    def distance(self, other):
        # some other method
        # need to use getter for `other`
        let dx = self.x - other.get_x()
        let dy = self.y - other.get_y()
        (dx * dx + dy * dy).sqrt()
Using the Posn class

```rust
define Posn(x: int, y: int):
    x: int
    y: int

let p = Posn(3, 4)
assert_eq p.get_x(), 3
assert_eq p.get_y(), 4
assert_error p.x  # fields are private

let q = Posn(0, 0);
assert_eq p.distance(q), 5
```
Discuss with your Neighbor

- Now that we’ve seen more of DSSL2, let’s repeat the exercise
- In 2 minutes, let’s hear your questions
Let’s look at a rational number class
For more DSSL2 information

See the DSSL2 reference (or help desk)
Next time: The humble linked list