* Control Flow
* Expressions
* Assignment

* Lab2 is posted.

10/8

Control Flow

How computation progresses through the program.
Every PL provides several mechanisms

- Expressions
- Sequencing
- Selection
- Repetition/Iteration
- Procedures/Functions
- Recursion
- Concurrency
- Exception Handling
- Non-determinacy.

Expressions

- literals - are values
  e.g. 32767 (int), True (boolean), -4.02 (float), "Hello" (string), 'C' (char)
- basic expressions
  e.g. a+b, a%b, a<=b, -a, !a, a||b
  - use operators: +, %, <=, -, !, ||
  - operands: a, b, or literals

Types of Expressions/Operators

1. Unary : <operator> <operand>
   e.g. -a, !a
2. Binary : <operand1>, <operand2> <operator>

Types of Binary Operator Expressions

Infix : <operand1> <operator> <operand2>

e.g. a+b, a%b, a<=b, a||b

Prefix : <operator> <operand1> <operand2>

		LISP
e.g.	+ a b	(+ a b)
	% a b	(% a b)
	<= a b	(<= a b)
		(* (+ a b) c) ≡ (a+b)*c

also postfix ---

Operator Precedence Rules

C

++, --	post e.g. i++, i--
++, --	pre ++i, --i
-	unary
!	unary
*, /, %	
+, -	
<, <=, >, >=	
==, !=	
&&	
||	
=, += ...	

Java

same as C

Python

**	x**y = x^y
-	unary
*, /, %	
+, -	
<, <=, >, >=	
==, !=	
not	
and	
or	
lambda	

why := missing??
In Python, we have :=

( ) can be used to override.

e.g. a+b*c ≡ a+(b*c)
(a+b)*c ≡ (a+b)*c

Evaluation order

- Left Associative: $a + b + c$ $\longrightarrow$
- Right Associative: $x ** y ** z$ ex. in Python

is it ~~$(x ** y) ** z$~~ or $x ** (y ** z)$

$(x^y)^z$ $\quad$ x^{y^z}

Most languages are ~~right~~ left-associative

Conditional operator / Expression (Ternary Operator)

?:

<expression> ? <expression> : <expression>

e.g. for

```
if (a > b)
    max = a
else
    max = b
```

```
max = a > b ? a : b ;
```

C, C++, C#, Java, JS, swift : `max = a > b ? a : b ;`

Python : `max = a if a > b else b`

Haskell `max = if a > b then a else b`

LISP `(setf max (if (> a b) a b))`

Rust `let max = if a > b {a} else {b}`

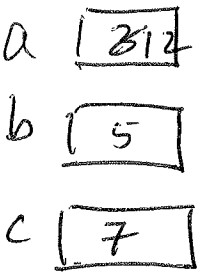
In imperative PLs computation is carried out via expressions and assignment statements

Assignment

<var> = <expression> a = b + c
<var> ← <expression> a ← b + c
<var> := <expression> a := b + c

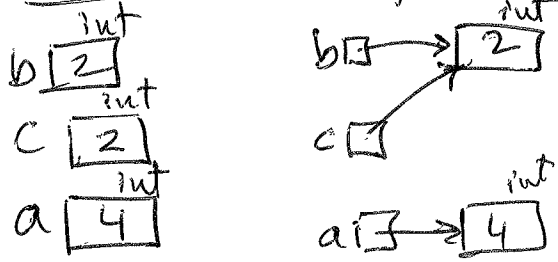
RHS: var name refers to its value — r-value
LHS: var name refers to its location — l-value.

e.g. a = b + c
a: mem location, l-value
b, c: 5, 7 r-values



Two Models of Variables: Value Model Reference Model

```
int b = 2;
int c = b;
int a = c + b
```



Value-Model: variable name is a named container for its value.

Reference Model: Variable name is a named reference to its value. Every name is an L-value

Java
- uses value model for all built-in types: int, long, float, double, boolean, char
- uses reference model for all class instances/objects: string, arrays