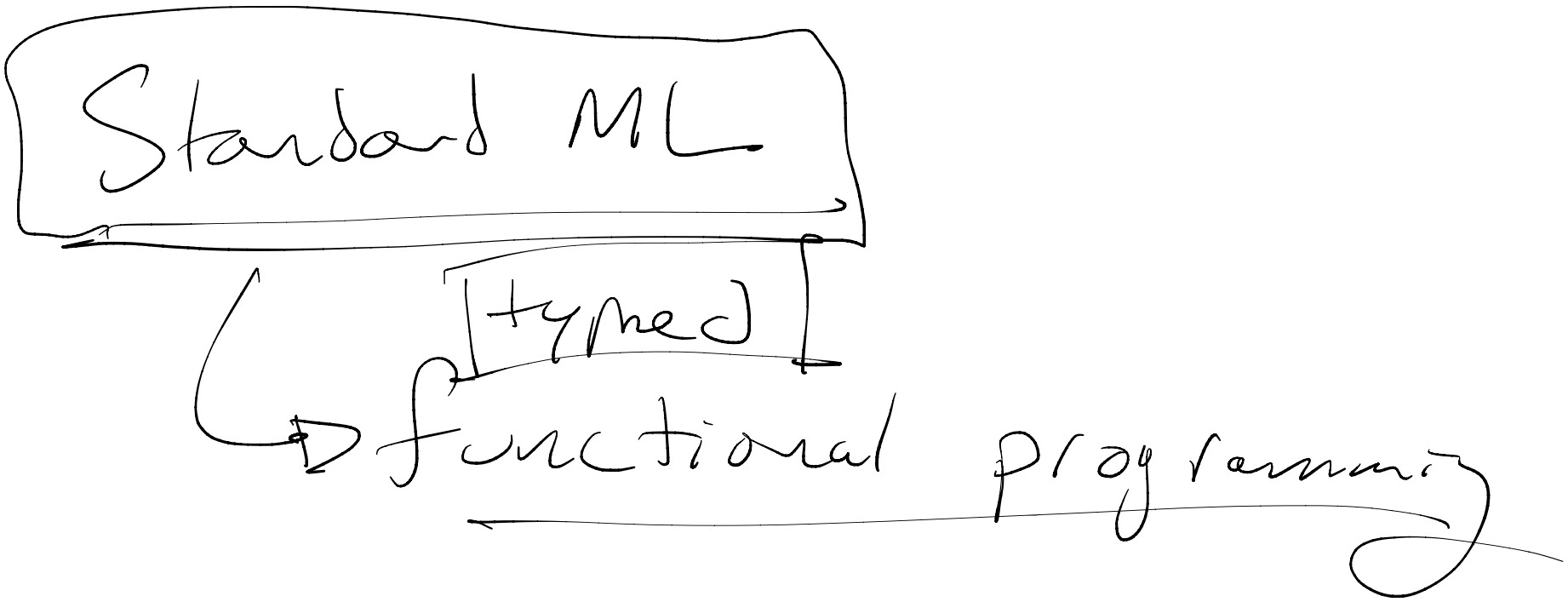
Lecture 2: Computing by calculation + SML basics



Expression	has a type	Answer / Value
2	int	2
1+1	int	2
(1+2) * (3+4)	int	21
"a"	string	"a"
"a" ^ "b"	string	"ab"
intToString 5	string	"5"
"a" + 1	ill-typed	(no value)
5 div 0	int	(no value)

$(1+2)*(3+4)$

$\longmapsto$ $3 * (3+4)$

steps to

$\longmapsto$ $3 * 7$

$\longmapsto$ 21 ✓

Sequential

$(1+2)*(3+4)$

$\Longmapsto$ $3 * 7$

parallel steps

$\Longmapsto$ 21

Parallel

Every type has a collection of values and operations

Int

values 0, 1, 2, ~1, ~2

operations + * div
IntToString ...

String

values "a" "b" "ab"

ops ^ ...

Type checking

(1) each of the values has the indicated type

(2) each operation is well-typed when the subexpressions have the right type

E.g. $(3+7)*5 : int$ because

$(3+7)*5$: expression

int : type

$3+7 : int$ because

$3 : int$ ✓

$7 : int$ ✓

$5 : int$ ✓

"a" + 1 :? int b/c

~~"a" : int~~ "a" : string

I : int

type error

↳ SML at "compile-time" before "run-time"

5 div 0 : int if

5 : int ✓

0 : int ✓

don't have

(int | not = 0)

Exceptions → reporting errors at run-time

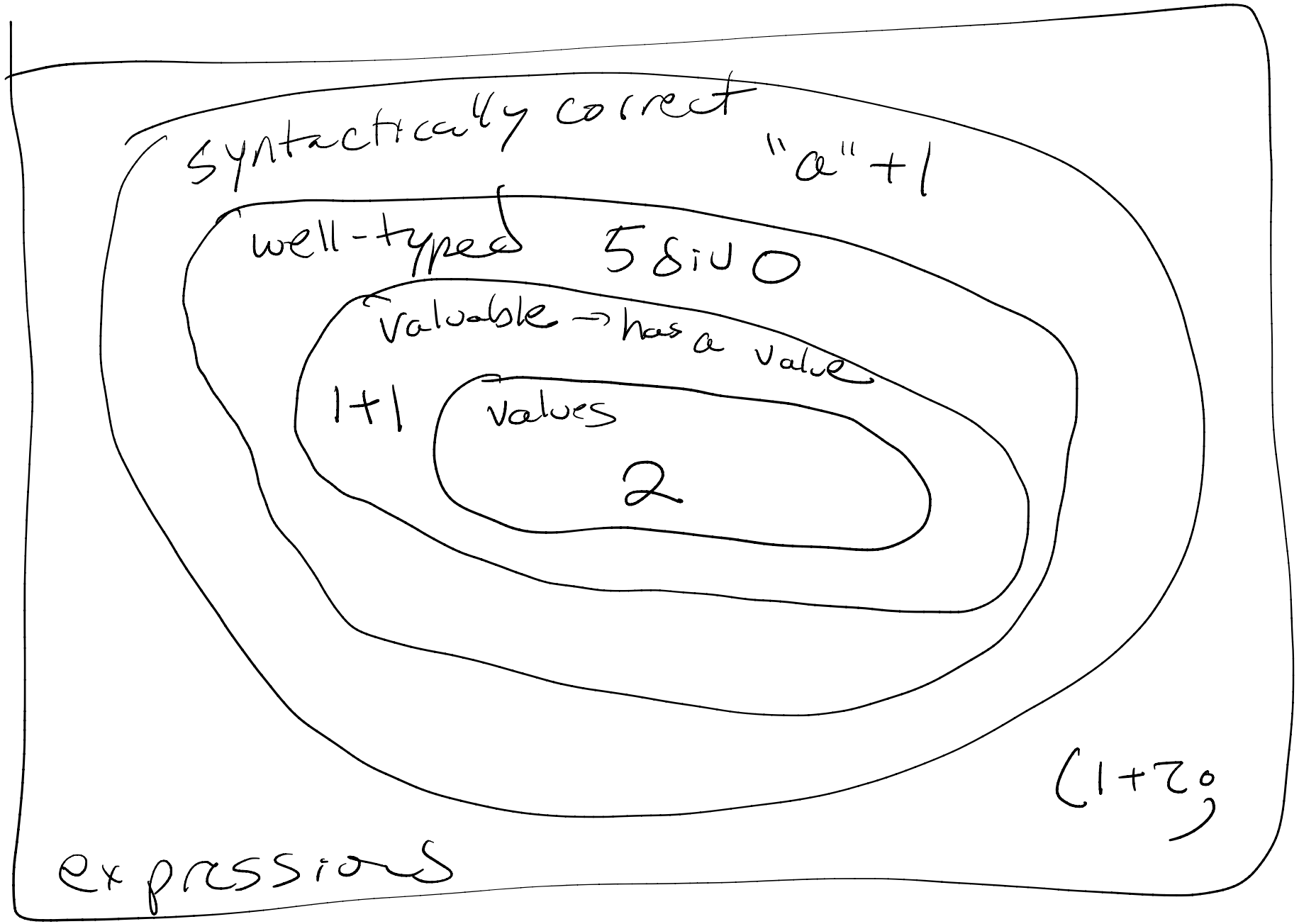
5 div 0

⟼ raise Div

↑ exception

5 div ((x * y) + z) − w

= 0 ?


expressions
syntactically correct
"a" + 1
well-typed
5 & 10
valuable → has a value
1+1
values
2
(1+2;

real

floating point number

(fixed precision 64 bit)

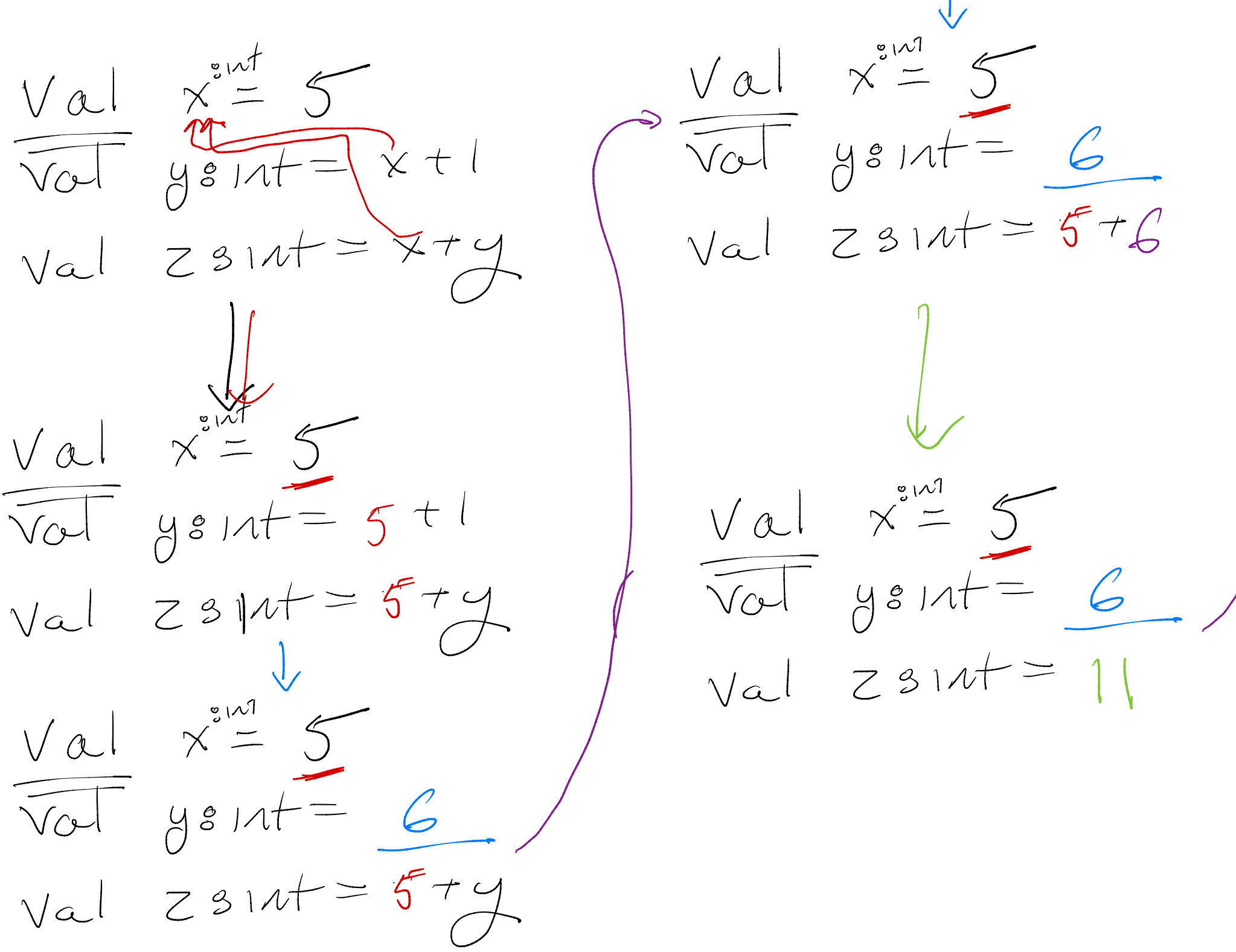
Values 1.0 1.1 3.14 ~ 2.1 ...

operations + * /

Variables: placeholder that stands for a value

```
val x:Int = 2+3
val y:Int = x+1
val z:Int = x+y
```

- body must have the indicated type
- variable has that type below


val x : int = 5
val y : int = x + 1
val z : int = x + y
val x : int = 5
val y : int = 5 + 1
val z : int = 5 + y
val x : int = 5
val y : int = 6
val z : int = 5 + y
val x : int = 5
val y : int = 6
val z : int = 5 + 6
val x : int = 5
val y : int = 6
val z : int = 11

In functional programming,

Variables

don't

vary

```
val x:int = 5
val y:int = x+1      6
val x:int = 3
val z:int = x+1      4
```

shadowing

variable uses refer to the nearest enclosing binding

```
val x:int = 5
val y:int = x+1
val w:int = 3
val z:int = w+1
```

Functions → capture patterns of computation for use on different inputs

Math $f(x) = 2x + 6$

SML

```
fun f (x : int) : int =
    2 * x + 6
```

name — input — type of the input — type of output — body

```
fun f(x: int) : int = 2 * x + 6
```

- Variable for input is assumed to have the specified type.

- body must have the specified type.

type checking

2 * x + 6 : int b/c

2 * x : int b/c

2 : int

x : int

6 : int

```
fun f(x: int) : int = (2 * x) + 6
```

Evaluate a function on an input

f 2 f 7

1. evaluate the argument to a value
2. substitute that value into the body of the function

f 2 "function application"

f 2

$\mapsto (2 * 2) + 6$

$\mapsto 4 + 6$

$\mapsto 10$

f 7

$\mapsto 2 * 7 + 6$

$\mapsto 14 + 6$

$\mapsto 20$

f 10

$\mapsto (2 * 10) + 6$

$\mapsto 26$

$f(2+2)$

$\mapsto f\ 4$

$\mapsto 2*4 + 6$

$\mapsto 8 + 6$

$\mapsto 14$

call by value

↳ SML

$f(2+2)$

$\mapsto (2*(2+2)) + 6$

$\mapsto$ ⋮

call by name [need]

↳ Haskell

is ill-typed

function application

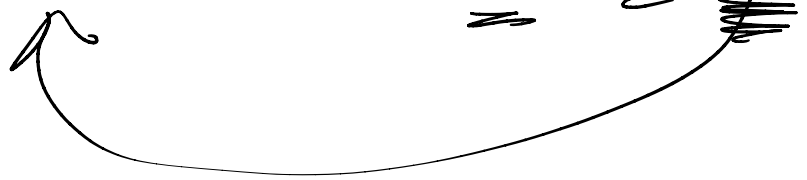
has the result type of the function if the argument has the indicated input type.

```
fun (x:int)+(y:int):int = ....

fun (x:string)^(y:string):string = ...

fun intToString(x:int):string = ...
```

built-in

fun f(x:int) = 2x+6

fun g(x:int) = f(f x)



Val y = x+1 (no x)

g(7)

↦ f(f 7)

↦ f(2*7+6)

↦ f(20)

↦ 2*20+6 ⟶ 46

expressions → values

↳ types

declarations of variables

functions