Type Checking: when are two types identical

$v_1 : t_1$
$v_2 : t_2$

1. $v_1 == v_2$?
2. $--- = v_1 + v_2 - - -$?
3. $f(v_1)$?

def $f(v : t_2)$ {
|
3

- Strongly + Weakly typed languages
- Static vs dynamic typing
- Type Equivalence — Name Equivalence
  \ Structural Equivalence
- Type Compatibility - a more practical approach

  Two types are compatible if

  1. They are equivalent
  2. One is a subtype of another
  3. Both are arrays w/ same type elements

JAVA Type Compatibility

1. Identical types
2. Subtypes + Supertype
3. Interfaces
4. Type Casting

Arrays

In some languages the index range of an array is part of the type

e.g. PASCAL

```
type A = array [1..10] of integer;
type B = array [0..9] of integer;

var a : A;
    b : B;
```

are a + b equivalent?

how about

```
function max (x : A)
    =
    =
```

max (b); ??

only if

```
function max (x : array [1..10] of integer)
    =
    =
```

still cannot do max (b) !

So, Modula-2 used open arrays

```
FUNCTION max (x : array of integer)
```

now we can call max (a);
max (b);

but indexing in max will be x [0..].

What about C + Java?

```
int max(int a[]) {
```

3

~~in the call call~~

we can call max() with any int array of any size.

BUT, how do we know how many elements in a[]?

e.g.

```
int [10] a;
int [100] b;

__ = max(a);
__ = max(b);
```

how does max know sizes of a[] + b[].

we have to do

```
int max(int [] a, int n) {
```

3

+ call

```
__ = max(a, 10);
   = max(b, 100);
```

What about Java?

```
public static int max(int a[]) {
```

— a.length is # elements in a[]

3

No need to send 'n' since Java uses the length attribute

Python!? len(a)

Type Conversion/Coercion

when two types may not be structurally equivalent but there is a way to convert one fm another.

— can be done implicitly or explicitly

e.g.
```
int x;
double y;
x=3;
y = y + x;
```
or `y=x;` ← x is implicitly converted to double.

however, doing `x=y;` does not make sense. why?

PLs provide either casts

i.e. `x = (int) y;`

or ~~`x = round(y);`~~

also, `int n = (int) (Math.random() * 10);`

`Math.random()` → [0.0 – 1.0) → 10.0

and in Java `System.out.println("A=" + 3);`

`3` → `"3"` → `"A=3"` → printed.

Python - Duck Typing !!

"If it talks like a duck, walks like a duck then it is a duck!"

- No explicit type compatibility rules
- Not ~~type~~ a strongly typed language

Python is dynamically typed.

Rule

If an object can perform the actions that are expected in a context, then it is compatible.

- int is compatible with float

errors happen at run time.

e.g. a = 5
len(a) ← a not a list!

Type Inference

- automatic detection of types of an expression

Javascript

$x := 5.3$ infered as float.

Haskell, SML, Javascript, etc.

Can get very complicated [where the PL theory research action is!]

e.g.
```
add_one(x){
    int result
    result = x+1
    return result.
```

inference

add_one : int → int

Haskell

map(double, a)

double :: int → int

a : [int]

∴ map :: (int → int) → [int] → [int]

More abstractly

```
map (f, l)
map f [] = []
map f (first : rest) = f first : map f rest
```

∴ map :: (α → β) → [α] → [β]

~~Arrays~~:

Control Abstraction - Subroutines

Control Abstraction vs Data Abstraction

Subroutines

- Functions : return value(s)
- Procedures : do not return any value

e.g. Java

```
double area (double radius) {
    return Math.PI * radius * radius;
} //area()

void drawCircle (double x, double y, double radius) {
    ...
    return;
    ...
    return;
} //drawCircle()
```

Functional

Vocabulary + Design Considerations

- caller, function call, invocation – double a = area(r);
- signature – double area(double)
- Arguments/actual parameters, formal parameters

  a = area(r); ← actual parameter

  show formal parameter in example code.
- return ——→ return <expression>;
  - primitive type only?
  - can be an aggregate type?
  - can be a function?
  - can return 0 or 1 value?
  - can return multiple values?
- Parameter Association (x, y, radius)
  - positional — drawCircle(x1, y1, r1);
  - named/keyword parameters

    drawCircle(x=x1, y=y1, radius=r1);

    drawCircle(radius=r1, y=y1, x=x1);
  - Mixed association

    drawCircle(x1, y1, radius=r1);
  - default parameters

    def convert(n, base=2):

    ———

    binary = convert(n)

    octal = convert(n, 8)

    hexadecimal = convert(n, base=16)
  - variable number of arguments

    printf(<string>, v1, v2, ...);

C

```
int max(int nargs, ...) {
    int max, a;
    va_list args;
    va_start(args, nargs);
    max = va_arg(args, int);
    for (int i=2; i<=nargs; i++)
        if ((a = va_arg(args, int)) > max)
            max = a;
    va_end(args);
    return max;
} //max()
```

```
max(5);
max(5, 1) → 5
max(5, 16, 7) → 16
```

Java

```
int max(int a, int ... ns) {
    int m = a;
    for (int x : ns)
        if (x > m)
            m = x;
    return m;
}
```

Python

```
def max(a, *argv):
    m = a
    for x in argv:
        if x > m:
            m = x
    return m
```