Topic 8: Types
Ch 7 Scott

2 basic questions: what / why
What??
bits are untyped!!!
most basic: a type defines how many, and how, to interpret bits. Similarly, in any
language, if a string is a “basic” type, how because you do not know its size
also—the set of operations that are allowed it.
primitive types “built in” — usually at hardware level
different from Java int, …
composite types

Why?:
1. Types supply context — Useful for compiler as it specified what to do
2. Limit what is allowed to be done
3. Make the program more readable to user — effectively a form of documentation —
especially useful when there are a lot of types (OO langs). So why type inference (as in Go/
Rust this seems to defeat self documenting)?
4. Compile time optimization
Most of these are arguments in favor of static types, What about languages (python) with
dynamic types point 2 is still valid.

Type system:
1. mechanism to define types
2. Definition of
   type equivalence
   structural vs name
   type compatibility
   what is allowed with what
   for + suppose one is Int, what is the other allowed to be
      in a weakly typed anything
      Go, Java, Rust
   type inference (may not be available in some langs)
Terms
static vs dynamic type
Python is dynamically typed.
is Javascript???
strongly typed
See below

So what is type in python?????
“Python’s dynamic typing is closely related to the concept of duck typing. Duck typing
emphasizes an object’s behavior over its class or type. In other words, the suitability of an
object for a particular operation is determined by whether it supports the required methods or
attributes, rather than checking its explicit type.”
https://medium.com/@mycodingmantras/understanding-the-dynamic-typing-nature-of-python-
a-comprehensive-guide-8f825fda0d01

Python — it seems as if variables don’t have types:
However, internally a has a type - it is PyObject* and this reference can be bound to an integer (1) and to an unicode-string ("1") - because they both "inherit" from PyObject. (As in java, each object knows what it was created as.)

So the interpreter infers the types during the run-time, but most of the time it doesn't have to do it - the goal can be reached via dynamic dispatch.

"primitive types" vs composite types
composites in next chapter
  struct, array, set, pointers, list, file
Primitive — int (at what precision?) should a lang care about precision?
  character? ASCII, 16-bit ascii? rune? UTF-8
enums — primitive or composite.
  lets say they are primitive but come back to in a few minutes

Do functions have types?
  Why?
  If they are first or second class, they do / must
What is the type of function??
  Go:
    type af func(a int) int
    func(increment int) int { return aa + inc }
  Rust
    much the same as Go
  Java— function type is its name and all of the types of its arguments
  do we even need to talk about function types in Java?? if not, why?

Strongly typed — language prohibits even trying to do something that is not allowed for a type.
Thrown out at compile
Weak—usually implies doing more work at run time — strong==fast
  for instance, to make the "+" work, javascript must do what?
can interpreted language be strongly typed?
  realistically this is a spectrum. Language may have holes ...
weakly typed — ex language allows application of operators when it does not make
  necessarily make sense. For instance, javascript is weakly typed (and dynamically typed)
  $f = $some function
  $q = 5 + f$
  Go? Rust?
  how does type coercion factor in here??
  does type coercion make language weakly typed??
  java has coercion — rust and go do not ... why not?

Statically typed — strong AND type checking is a compile time.

Lots of types
Basic type: integer, float …
Integers
Java: byte, short, int, long. Also, Byte, Short, Integer, Long, BigInteger!!!
Rust [u,i][8,16,32,64,128, size]
Go: [u]int[8,16,32,64]
Why so many int types???
Floating point: similar
  go and rust f32, f64,
char — what is a char?
  one byte — ASCII
  char in c
  2 bytes — UNICODE16 — JAVA
  char in Java
  rust “The char type represents a single character. More specifically, since
  ‘character’ isn’t a well-defined concept in Unicode, char is a ‘Unicode scalar value’. … USVs
  are also the exact set of values that may be encoded in UTF-8. All USVs are valid char values,
  but not all of them represent a real character. Many USVs are not currently assigned to a
  character,” from the rust book
  Go does not actually have a char type it has a “rune”
WHAT IS A RUNE IN GO?
  Up to 4 bytes — UTF8 — variable
  0xxxxxxx — 1 byte — plain old ASCII
  110xxxxx 10xxxxxx —
  1110xxxx 10xxxxxx 10xxxxxx
  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

is String a basic type?
  in Java? C? Go?
  Java — NO..it is a class
  (Are classes in java.lang really “basic” to Java??
  You cannot do ANYTHING without java.lang.Object
  To know would have to look at implementation of String class
  C — definitely NOT
  Go — from book “a string contains an array of bytes that, once created,
is immutable”
    This indicates that string is a composite type, maybe
    Going further Go explicitly mirrors string functions with byte array
functions
  boolean, string or number"
  Rust —
    “String literal”? &str
    String — NO — the rust book says it is really a vector

Enumerated types
What: a type that has a specific, finite (usually small), and bounded set of
possible values.
  Why????
  How is this encoded by the language …
  consecutive integers? Powers of two?
see enum_java/GTEnum.java

Why?

Go: enum_go/enum.go
They do not really exist like in other languages so you get little benefit
an they are certainly not primitive

Java: enum_java/GTEnum.java
Rust: enum_rust/src/main.rc

Type checking
Java: obvious and handled by compiler
Go: often do not require explicit types (type inference)
type inference
why have type inference?
you loose the readability of the implicit documentation
what do you gain?

When are two types the same???
structure vs name equivalence
structural
same order, or just same number and kind?
what work needs to be done to get this?
what does Go/Elixir do?
why not use structural equivalence?
name
what about type aliases?

what are Go, Java
Go: equiv_go/equiv.go
strict name equivalence
NOTE: structural equivalence is about does the question of equality even make sense?
Should the question even be allowed?

Java: no type aliases (quite) equiv_java/Equiv.java
you can define a class that extends another class without addition
Why would you??
limitation — class cannot be final (e.g. String is final, why?) what
is final with respect to classes in Java?
Also this does not really get you equivalence

Rust —
has type aliasing but the aliases seems to be taken out at compile
time??
are structs a type??
effectively yes.

Casting — converting from one type to another
in strongly typed languages “weird” casts are not allowed
GO: casts_go/casts.go
func t5() {
str := "abc"
fmt.Println(str)
var num int64
num=40
fmt.Println(num)
num = int64(str) // Compiler flags as not allowed
}

Problem is that casting requires changing bits and you have to know how.
what is the problem with changing bits?? time!
Some langs allow “non-converting” casts. That is, do not change bits just interpret
bits differently. What is problem? (C does this. Why?) nonconvert_c/pun.c
rust can do it nonconvert_rust/src/main.rs
Go: pun_go/pun.go
uses a package named “unsafe”

Question — can you do this in Java?? Why/why not??

type coercion
implicit casting????
allow 3+2.4 without explicit casing
pros/cons
Go — no coercion
Java — happy to coerce among numeric types
Javascript — (weak) happy to coerce pretty much anything
— “JAVASCRIPT WANTS THINGS TO BE TRUE”
== vs === in javascript

Type inference (in statically typed language):
go does it:
infer_go
type inference in Java??
does <> in some generics count as type inference??

Advantages / disadvantages of type inference (in a strongly typed language)???

Generics
they are much more complex that you thought (and you probably thought they were
pretty complex)
Java “Generic Gotchas”
See the web article
Covariance & Generics:
For example
Integer extends Number — True
By Covariance Integer[] extends Number[]
Hence this is legal:
Number[] nArray = new Number[10];
Integer[] iArray = nArray;
can put integers into iArray and it is guaranteed to be fine with
nArray

See **ArrayCov java**
point when passing into methods covariant type inherit just like
their base types. But this can cause issues at run time.
generics are NOT covariant It would break type safety
For instance consider ArrayList
ArrayList<Integer> ai = new ArrayList<>();
ArrayList<Number> an = ai; // WILL NOT COMPILIE
ln.add(Double.doubleValue(2.2));

See also **Cov1 java**
(note arrays actually have the same issue)

Generics with wildcards
see covar_java
see Wildcard java
ArrayList<? extends Number>
ArrayList<?>
ArrayList<*>
Wildcards can be handy
limit a function to taking an array list that contains anything that extends
number (you need it here because generics are NOT covariant)
But wildcards result in other issues, specifically immutability.

See **Immut java**

Type erasure in Java
generics are known only by compiler, they are “erased” after compile so all of
that info is gone at runtime.

see **Erasure java**

EG
ArrayList<String> ss = new ArrayList<>();
eventually gets translated to
ArrayList ss = new ArrayList();
So at run time, anything that the compiler let pass is OK. It could cause runtime
issues.
Erasure also causes things that might see legal to NOT be legal. For instance
public class JavascriptNumber implements Comparable<String>,
Comparables<Number> { ...}
does not work because compiler reduces this to
public class JavascriptNumber implements Comparable, Comparable { ...}

Generics in Go
See **GoGen1** for basics
NO erasure in Go … see **GoGen2**
Any — kind of like Object in Java. More like ?
LinkedList is a good example, but not until next chapter!

Object equality (sec 7.4)
deep vs shallow equality
deep vs shallow assignment
in ref-model and value model languages
why in Go if == defined over array but not slice
“deep assignment”

When are two objects the same?
Deep vs shallow checks?
Java == vs equals
    Deep vs shallow assignment
Only applied to reference model languages
    see copy.go
Value languages effectively always deep copy
Shallow
    copy and assign pointer (SCopy.java)
    make a new copy of object and assign.