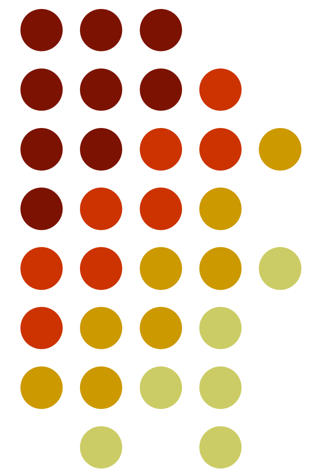
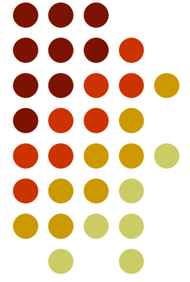
Variables

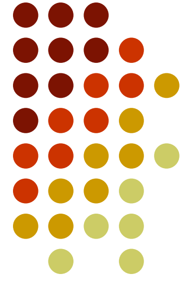
CS181: Programming Languages





Topics:

- Static vs. dynamic typing
- Strong vs. weak typing
- Pointers vs. references



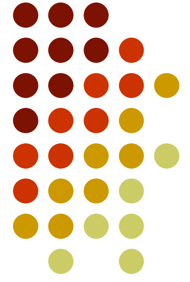
Static typing

- Variables are bound to their type at compile time, and cannot be subsequently changed. This binding is (typically) specified by variable declaration. For example, in C:

```
int foo = 2;
int bar = 3;
bar = bar + foo;
```

- C, C++, Java, Pascal, Ada, ML are statically typed.

Dynamic typing

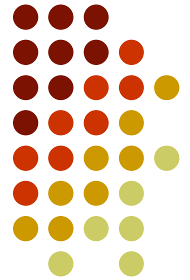


- Type of the variable depends on the value dynamically (run-time) associated with it. For example, in Python:

```
foo = 2
bar = 3
bar = foo + bar
```

- Python, PHP, Perl, Ruby, Scheme, Lisp, Basic, Smalltalk are dynamically typed.

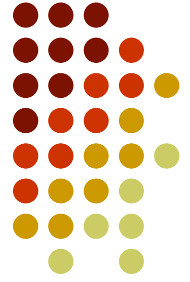
Polymorphic variables vs. OO polymorphism



- Dynamically typed variables are also called *polymorphic* variables (ancient Greek: *poly* = many, *morphos* = shape).
- Dynamic typing is however not to be confused with polymorphism in the OO sense. (reminder: polymorphism is the ability of objects belonging to different types to respond to methods of the same name). For example, in Java:

```
Object A;
A = new String("foo");
A = new Float(2.718281828);
A.toString();                will work
A.floatValue();              will not work
```

Casting

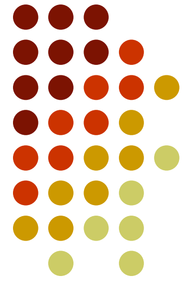


- Dynamic typing is not to be confused with casting. For example, in C:

```
int foo = 2;
float bar = 2.718281828;
bar += (float) foo;
```

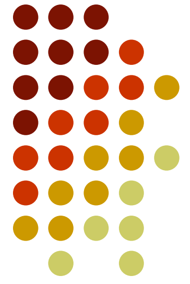
- Casting in this case does not change the type of foo into float. It only allows you to read an int variable as a float variable.

Static vs. dynamic typing



- Static typing is (usually) associated with compiled languages.
- Dynamic typing is (again, usually) associated with interpreted or scripting languages.
- Typing does not only influence when the variable is bound – it also influences when type-checking will happen.

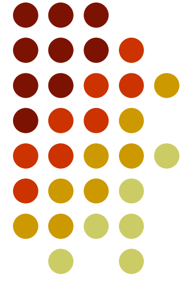
Static vs. dynamic typing



- Dynamic typing is friendlier (less code, simpler).
- Static typing catches more errors at compile time (and it is always beneficial to detect programming errors as quickly as possible); also, code is faster (compiler knows more and it can optimize).

- Each has its own benefits and shortcomings (i.e. trade-offs are a recurring topic in programming language design)

Strong typing



- Strong typing means strict enforcement of type rules with no exceptions. In a strongly typed language, conversion between types has to be *explicit*. For example, in Python:

  foo = “x”

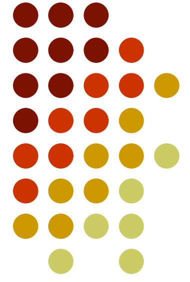
  foo = foo + 2

- results in an error. However:

  foo = foo + str(2)

- will work.

Weak typing



- Weak typing means that there are exceptions in the strict enforcement of type rules. For example, in C/C++:

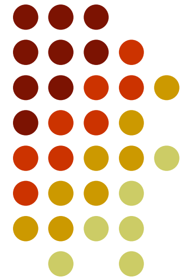
  ```
  int a = 5;
  float b = a;
  ```

- int was automatically coerced into float. Or, in Perl:

  ```
  print "1"+12
  print "1".111
  ```

- Informally, weak typing means that you can mix types without an explicit conversion.

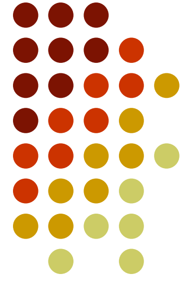
Static/dynamic/strong/weak



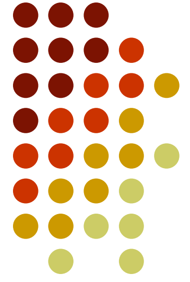
- Static vs. dynamic typing is completely independent from strong vs. weak typing.
- All combinations are allowed. For example:

	static	dynamic
strong	Java, Ada, Pascal	Python
weak	C / C++	BASIC, PHP

Pointers



- store a memory location of an object (variable, array, instance of a class, function, etc.)
- can be used to access the content of whatever they are pointing to.
- are not permanently attached to an object.

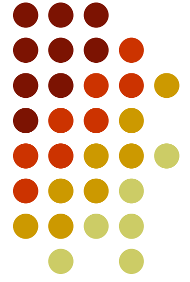


Pointers

- Sample C code:

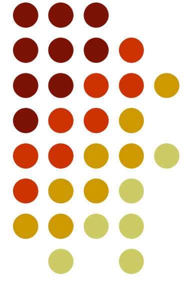
```
typedef struct dot_  {
        int x, y;
} Dot;
…
int main(int argc, char* argv)  {
        Dot *d;
        d = (Dot *) malloc(sizeof(Dot));
        d->x = 12;
        d->y = 8;
}
```

References



- are permanently attached to objects.
- unlike pointers, references have to be initialized at the point of definition.

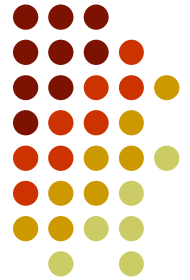
References



- Sample Java code:

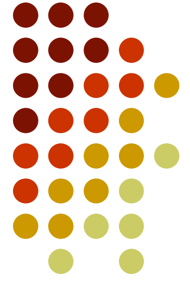
```
public class Dot {
        int x, y;
        …
        public static void main() {
                Dot d;
                d = new Dot();
                d.x = 12;
                d.y = 8;
        }
}
```

Pointers vs. references



- Using pointers results in faster code.
- However, great many errors arise from using pointers improperly. Think of, for example:
  - uninitialized pointers
  - pointers to deallocated memory
  - pointers beyond the end of an array
- References allow for garbage collection
- Pointers allow for memory leaks

References:



- Ghezzi C., Jazayeri M. *Programming Language Concepts*. 3rd ed. John Wiley and Sons. 1998.