02 Marty Stepp CS 106X: Programming Abstractions, Autumn 2017 Lecture 02-Functions

Namespaces and using

- **namespace**: An area for scoping identifiers (functions, variables).
  - Helps avoid collisions between items with the same name.
  - C++ console I/O objects (cout, cin, etc.) are in name space `std` .
- `using namespace` ***`name`***`;`
  - Brings symbols from a library's "name space" into the global scope of your program so you can refer to them.
- ***`namespace`***`::`***`identifier`***
  - without a `using` declaration, you can access symbols from a namespace by preceding them with their namespace name and `::`.

```
std::cout << "Hello, world!" << std::endl;
```

9:23 / 51:29

Output parameters

xkcdDatingRange

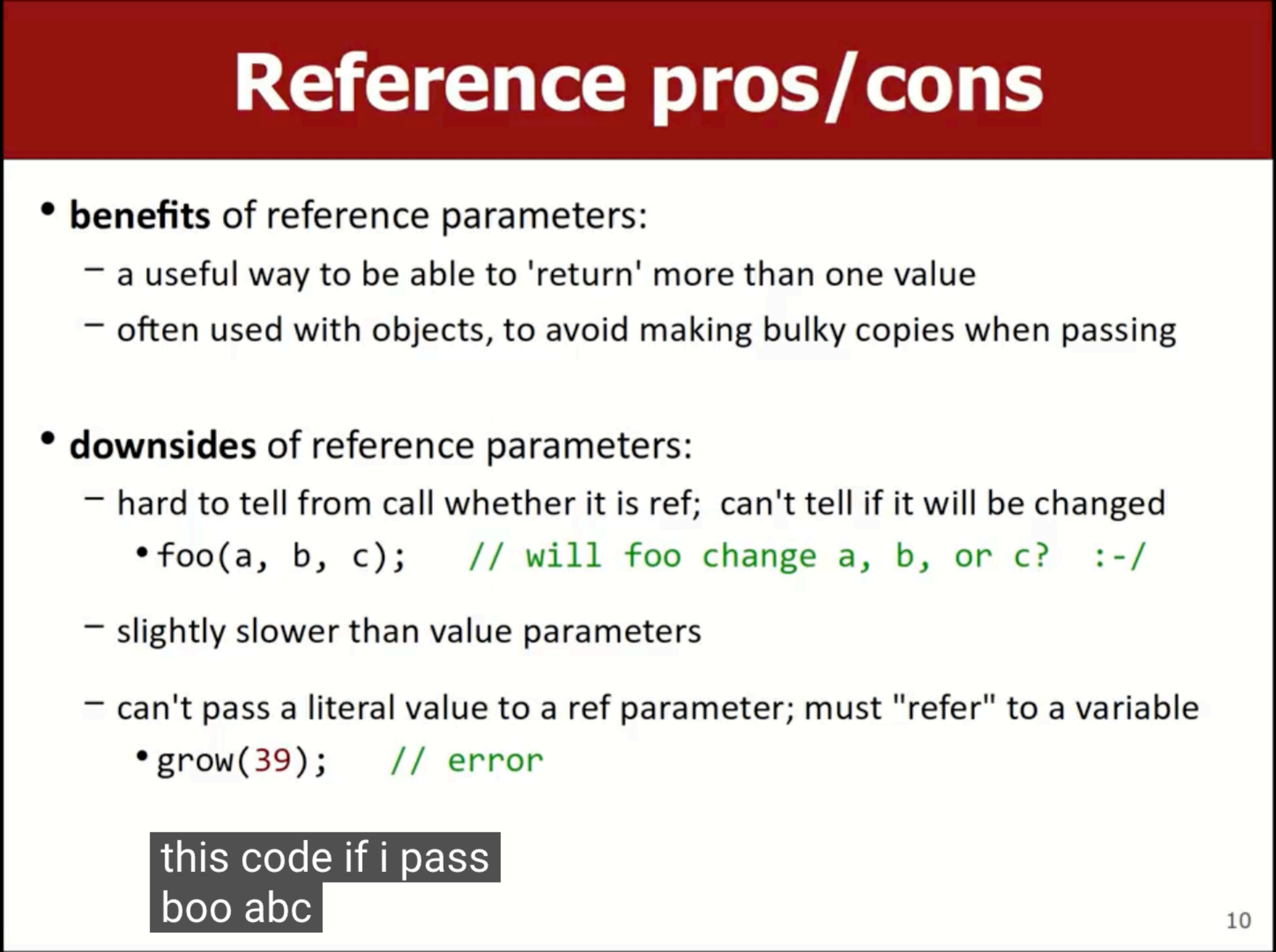
- What is the minimum and maximum non-creepy age to date?

```
void datingRange(int age, int& min, int& max) {
    min = age / 2 + 7;
    max = (age - 7) * 2;
}

int main() {
    int young;
    int old;
    datingRange(48, young, old);
    count << "A 48-year-old could date someone from "
          << young << " to " << old " years old." << endl;
}

// A 48-year-old could date someone from
// 31 to 82 years old.
```

YES, OLDER SINGLES ARE RARER. BUT AS YOU GET OLDER, THE DATEABLE AGE RANGE GETS WIDER. AN 18-YEAR-OLD'S RANGE IS 16-22, WHEREAS A 30-YEAR-OLD'S MIGHT BE MORE LIKE 22-46.
STANDARD CREEPINESS RULE: DON'T DATE UNDER (AGE/2 + 7)

http://xkcd.com/314/

design gets set up here
when min refers to young



Reference pros/cons

- **benefits** of reference parameters:
  - a useful way to be able to 'return' more than one value
  - often used with objects, to avoid making bulky copies when passing
- **downsides** of reference parameters:
  - hard to tell from call whether it is ref; can't tell if it will be changed
    - `foo(a, b, c);    // will foo change a, b, or c?  :-/`
  - slightly slower than value parameters
  - can't pass a literal value to a ref parameter; must "refer" to a variable
    - `grow(39);    // error`

this code if i pass
boo abc