

## Control Structures

---

---

---

---

---

---

---

### if-elsif-else

- semantically the same as C/C++
  - syntactically, slightly different.
- ```
if ($a > 0){  
  print "\$a is positive\n";  
} elsif ($a == 0){  
  print "\$a equals 0\n";  
} else {  
  print "\$a is negative\n";  
}
```
- brackets are *\*required\**!

---

---

---

---

---

---

---

### unless

- another way of writing `if (! ...) { }`
- analogous to English meaning of “unless”
- `unless (EXPR) BLOCK`
  - “do BLOCK unless EXPR is true”
  - “do BLOCK if EXPR is false”
- can use `elsif` and `else` with `unless` as well

---

---

---

---

---

---

---

## while/until loops

- while is similar to C/C++
- `while (EXPR) BLOCK`
  - “While EXPR is true, do BLOCK”
- `until (EXPR) BLOCK`
  - “Until EXPR is true, do BLOCK”
  - “While EXPR is false, do BLOCK”
  - another way of saying `while (! ...) { }`
- again, brackets are \*required\*

---

---

---

---

---

---

---

---

## for loops

- Perl has 2 styles of for.
- First kind is virtually identical to C/C++
- `for (INIT; TEST; INCREMENT) { }`

```
for ($i = 0; $i < 10; $i++){  
  print "\$i = $i\n";  
}
```
- yes, the brackets are required.

---

---

---

---

---

---

---

---

## foreach loops

- Second kind of for loop in Perl
    - no equivalent in core C/C++ language
  - `foreach VAR (LIST) { }`
  - each member of LIST is assigned to VAR, and the loop executed
- ```
$sum = 0;  
foreach $value (@nums){  
  $sum += $value;  
}
```

---

---

---

---

---

---

---

---

## More About for/foreach

- for and foreach are actually synonyms
  - they can be used interchangeably.
  - code \*usually\* easier to read if conventions followed:
    - `for ($i = 0; $i<10; $i++) {}`
    - `foreach $i (@array) {}`
- list argument of foreach can be any list
  - `foreach $i (@array) {}`
  - `foreach $i (1, 2, 3, 4, 5) {}`
  - `foreach $i (1 .. 5) {}`
  - `foreach $field (split /:/, $data) {}`

---

---

---

---

---

---

---

---

## Two More Thing... (about for)

- `foreach VAR (LIST) {}`
- while iterating through list, VAR becomes an \*alias\* to each member of LIST
  - Changes within loop to VAR affect LIST
- if VAR omitted, `$_` used

```
@array = (1, 2, 3, 4, 5)
foreach (@array) {
    $_ *= 2;
}
```
- @array now → (2, 4, 6, 8, 10)

---

---

---

---

---

---

---

---

## Loop Control – next, last, redo

- `last` → equivalent of C/C++ `break`
  - exit innermost loop
- `next` → (mostly) equivalent of C/C++ `continue`
  - begin next iteration of innermost loop
- `redo` → no real equivalent in C/C++
  - restart the current loop, without evaluating conditional

---

---

---

---

---

---

---

---

### continue block

- while, until, and foreach loops can have a continue block.
- placed after end of loop, executed at end of each iteration
- executed even if loop broken out of via next (but not if broken via last or redo)

```
foreach $i (@array)
  if ($i % 2 != 0){
    next;
  }
  $sum += $i;
} continue {
  $nums++;
}
```

---

---

---

---

---

---

---

---

### Breaking Out of More Loops

- next, last, redo operate on innermost loop
- Labels to break out of nesting loops

```
TOP: while ($i < 10){
  MIDDLE: while ($j > 20) {
    BOTTOM: foreach (@array){
      if ($j % 2 != 0){
        next MIDDLE;
      }
      if ($i * 3 < 10){
        last TOP;
      }
    }
  }
}
```

---

---

---

---

---

---

---

---

### goto

- yes, it exists.
- No, don't use it.

---

---

---

---

---

---

---

---