Loops in Pseudocode

Christophe Rhodes

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

$x \leftarrow 0$
for $0 \leq i < 100$ **do**
    $x \leftarrow x + 1$
end for

what is the value of x after this? 100

For loops

To loop with a variable bound to a series of numbers, use **for** with a description of the series.

$x \leftarrow 0$
for $0 \leq i < 100$ **do**
    $x \leftarrow x + i$
end for

what is the value of x after this? 4950

For loops

The order might matter: start with the left-hand bound and move towards the right-hand one.

```
x ← 0
for 0 ≤ i < 100 do
    x ← i
end for
```

```
x ← 0
for 100 > i ≥ 0 do
    x ← i
end for
```

what is the value of x after each of these?

99

0

For loops

Use **continue** to proceed directly to the next iteration of the innermost loop, and **break** to finish the innermost loop

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        break
    end if
    x ← x + i
end for
```

```
x ← 0
for 0 ≤ i < 10 do
    x ← x + 1
    if x > 3 then
        continue
    end if
    x ← x + i
end for
```

what is the value of x after each of these?

Nested loops

Loops nest: with nested loops, for each iteration of an outer loop, do the whole inner loop:

```
x ⟵ 0
for 0 ≤ i < 3 do
    for 0 ≤ j < 4 do
        x ⟵ x + 1
    end for
end for
```

what is the value of x after this? 12

Forall

Iterate over members of a collection using **forall**

```
x ⟵ 0
for all p ∈ prime numbers below 10 do
    x ⟵ x + 1
end for
```

what is the value of x after this? 4

ordering

There may be a natural order to the iteration (*e.g.* when iterating over a linear collection), but usually there won't be. Don't rely on a particular order!

While

Use **while** to express a loop which tests a condition at the start of a sequence, and if that condition is true does another iteration of the loop.

```
x ⟵ 0
y ⟵ 3
while y > 0 do
    x ⟵ x + 1
    y ⟵ y − 1
end while
```

what value does x have after this? 3

Repeat

Use **repeat until** to express a loop which tests a condition at the end of a sequence, and if that condition is false does another iteration of the loop.

```
x ⟵ 0
y ⟵ 3
repeat
    x ⟵ x + 1
    y ⟵ y − 1
until y < 0
```

what value does x have after this? 4

Loop

Use **loop** to express an unconditional loop. (You will need to use **break** to terminate the loop).

```
x ⟵ 11342
loop
    if x = 1 then
        break
    else if x is even then
        x ⟵ x ÷ 2
    else
        x ⟵ 3 × x + 1
    end if
end loop
```

Function calls

Functions have zero or more arguments, and return one result. Call them using their name, with arguments in brackets

$n \longleftarrow 5$

$x \longleftarrow \textsc{fact}(n)$

Functions

Define functions using **function**, and return a value using **return**.

```
function FACT(n)
    if n = 0 then
        return 1
    else
        return n × FACT(n−1)
    end if
end function
```

Pre- and post-conditions

Make it clear to the reader what conditions a function requires to operate correctly, and what it does if those conditions are met

Require: $n \in \mathbb{N}_0$
Ensure: Compute and return n!

```
function FACT(n)
    if n = 0 then
        return 1
    else
        return n × FACT(n−1)
    end if
end function
```