

CSCI 145 · Quiz 3 Reference Sheet (Lectures 5–6)

COMPARISON OPERATORS

Compare two values and produce a bool, True or False.

Operator	True when
<code>a < b</code>	a is less than b
<code>a > b</code>	a is greater than b
<code>a <= b</code>	a is less than or equal to b
<code>a >= b</code>	a is greater than or equal to b
<code>a == b</code>	a is equal to b
<code>a != b</code>	a is not equal to b

Careful: = assigns a value; == asks whether two values are equal.

LOGICAL OPERATORS

<code>a and b</code>	True only if both a and b are True
<code>a or b</code>	True if at least one of a, b is True
<code>not a</code>	True if a is False (flips it)

a	b	a and b	a or b
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Evaluation order (first to last): (), then arithmetic (** * / // % + -), then comparisons, then not, then and, then or.

CONDITIONALS: IF, IF/ELSE, IF/ELIF/ELSE

```
if condition:
    statements
```

Block runs only when the condition is True

```
if condition:
    statements
else:
    statements
```

Exactly one of the two blocks runs

```
if condition1:
    statements
elif condition2:
    statements
else:
    statements
```

- Conditions are checked **in order**; only the block for the **first** True one runs, and the rest are skipped
- You can chain as many **elif**s as you need; **else** is optional and must come last
- Separate **if** statements are each checked on their own, so more than one block can run
- The **:** ends each header line; indentation marks which lines are inside a block
- **return** ends a function immediately, even from inside a block

CONTROL FLOW

Sequence: statements run top to bottom · **Selection:** **if** chooses a path · **Repetition:** loops take the same path again and again

WHILE LOOPS

```
set up the loop variable
while condition:
    statements
    update the loop variable
```

- The condition is checked **before** each pass. If it is True, the body runs, then Python jumps back up and checks again
- If the condition is False the first time, the body runs **0 times**
- The loop ends when the condition is False
- The body must eventually make the condition False. If it doesn't, you have an **infinite loop** (in Thonny, click **Stop**)

FOR LOOPS

```
for variable in sequence:
    statements
```

- The body runs once for each item in the sequence
- Each pass, the loop variable holds the next item; the loop ends when the sequence runs out
- No condition to write and no update to forget

THE RANGE FUNCTION

Produces a sequence of whole numbers. The **stop value** is **never included** (same rule as slicing).

Call	Values
<code>range(stop)</code>	0, 1, 2, ... up to but not including stop
<code>range(start, stop)</code>	start, start + 1, ... up to but not including stop
<code>range(start, stop, step)</code>	start, start + step, start + 2·step, ... as long as they are less than stop

`range(2, 6) →`

2	3	4	5	6
---	---	---	---	---

To include n itself, the stop has to be one past it: `range(1, n + 1)`

FOR OR WHILE?

- **for** when you know how many times to repeat, or have a sequence to go through
- **while** when you repeat until something happens