```
(if question-expression                                    syntax
    then-answer-expression
    else-answer-expression)
```

When the value of the *question-expression* is `#true`, `if` evaluates the *then-answer-expression*. When the test is `#false`, `if` evaluates the *else-answer-expression*.

If the *question-expression* is neither `#true` nor `#false`, `if` reports an error.

```
(and expression expression expression ...)                 syntax
```

Evaluates to `#true` if all the *expression*s are `#true`. If any *expression* is `#false`, the `and` expression evaluates to `#false` (and the expressions to the right of that expression are not evaluated.)

If any of the expressions evaluate to a value other than `#true` or `#false`, `and` reports an error.

```
(or expression expression expression ...)                  syntax
```

Evaluates to `#true` as soon as one of the *expression*s is `#true` (and the expressions to the right of that expression are not evaluated.) If all of the *expression*s are `#false`, the `or` expression evaluates to `#false`.

If any of the expressions evaluate to a value other than `#true` or `#false`, `or` reports an error.

```
(check-expect expression expected-expression)              syntax
```

Checks that the first *expression* evaluates to the same value as the *expected-expression*.

```
(check-expect (fahrenheit->celsius 212) 100)
(check-expect (fahrenheit->celsius -40) -40)

(define (fahrenheit->celsius f)
  (* 5/9 (- f 32)))
```

```
(cond [question-expression answer-expression] ...)         syntax
(cond [question-expression answer-expression]
      ...
      [else answer-expression])
```

Chooses a clause based on some condition. `cond` finds the first *question-expression* that evaluates to `#true`, then evaluates the corresponding *answer-expression*.

If none of the *question-expression*s evaluates to `#true`, `cond`'s value is the *answer-expression* of the `else` clause. If there is no `else`, `cond` reports an error. If the result of a *question-expression* is neither `#true` nor `#false`, `cond` also reports an error.

`else` cannot be used outside of `cond`.

The design recipe

DATA	FUNCTIONS
① Data definition *Give the data definition a name, and state the set of values that are part of it.*	**① Signature** *Give the name of the function, the argument types that it expects as input, and what type it returns.*
② Interpretation *State how values should be interpreted, covering each field/clause if there are multiple of them.*	**② Purpose statement** *Describe in one sentence what the function does. This should be roughly the length of a tweet.*
③ Examples *Provide a set of representative examples, building up complex examples iteratively.*	**③ Tests** *Provide a set of representative tests, covering different kinds of input and different behaviors.*
④ Template *Provide a template for functions that accept this data definition as input.*	**④ Code** *Starting with the template for the input data definition, write the code for the function.*

```
(substring s i j) → string                                    procedure
  s : string
  i : natural?
  j : natural?
```

Extracts the substring starting at i up to j (or the end if j is not provided).

```
> (substring "hello world" 1 5)
"ello"
> (substring "hello world" 1 8)
"ello wo"
> (substring "hello world" 4)
"o world"
```

```
(string-append s t z ...) → string
  s : string
  t : string
  z : string
```

Concatenates the characters of several strings.

```
> (string-append "hello" " " "world" " " "good bye")
"hello world good bye"
```

```
(string-length s) → nat
  s : string
```

Determines the length of a string.

```
> (string-length "hello world")
11
```