Chapter Two

- expression - a fragment of code that produces a value
  - corresponds to a sentence fragment
  - simplest kind: an expression w/ a ; after it
  - can just produce a value, which can then be used by the enclosing code
- statement - corresponds to a full sentence
  - program = a list of statements
  - stands on its own ⟶ Δs = side effects
- bindings / variables - things that catch & hold values
  - keyword let indicates that a sentence is going to define a binding ⟶ followed by the name of the binding
    `let bindingName = expression;`
  - when a binding points to a value, it does not mean that it's tied to that value forever
    ↳ bindings = tentacles ⟶ they grasp them
  - 2 bindings can refer to the same value
  - the value of an empty binding = undefined
  - single let statement can define mult. bindings, but the definitions must be sep. by commas
  - other bindings:
    var: mostly the same as let
    const: constant ⟶ points @ the same value for as long as it lives
  - binding names:
    ↳ must not start w/ a digit
    - can incl. $ or underscores
- environment - collection of bindings & their values that exist @ a given time
  - is never empty ⟶ always contains bindings that are part of the language standard & (most of the time) has bindings that provide ways to interact w/ the surrounding system
- functions - a piece of program wrapped in a value; such values can be applied in order to run the wrapped program
  - executing a function = called invoking
    calling
    applying it
    can call a function by putting parentheses after an expression that produces a function value
  - arguments - values given to functions
- console.log - writes out its arguments to some text output device
  ↳ actually an expression that retrieves the log property from the value held by the console binding
- return values = a side effect
  - when a function produces a value, it is said to return that value
- control flow - statements = executed from top to bottom
- conditional execution - created w/ the if keyword
  - if: executes or skips a statement depending on the value of the Boolean expression
  - Number.isNaN: standard JS function that returns T only if the argument given is NaN
  - block - grouping of any # of statements into a single statement
    - wrapped in {}
- loops:
  - while - loops keeps entering the statement as long as the expression produces a true value
  - do - like a while loop, but always executes its body @ least once and then it starts testing whether it should stop only after that first execution
- w/ proper indentation, the visual shape of a program corresponds to the shape of the blocks inside it

- for loops:
  1. counter binding = created to track the progress of the loop → initializes
  2. while loop — usually w/ a test expression that checks whether the counter has reached its end value → checks
  3. counter = updated to track progress → updates
- break - special statement that has the effect of immediately jumping out of the enclosing loop
- continue - also infl. the progress of a loop
  ↳ when encountered in a loop body, control jumps out of the body & continues w/ the loops next iteration
- switch statement: a direct way to execute a chain of if statements

```
switch (___) {
    case ___ :
      _____;
      break;
    default :
      _____,
      break
}
```

- binding names = camel Case
- constructor names = 1st letter capitalized