Functions

Ref Javascript

Parameters & return

A function is a reusable piece of code. A function declaration gives it a name, which you can use to call it:

```
function greet(name) {
  return `Hello, ${name}!`;
}

console.log(greet("Mei"));   // Hello, Mei!
console.log(greet("Tom"));   // Hello, Tom!
```

- `name` is a parameter : a variable that receives a value when the function is called.
- `"Mei"` is the argument : the actual value passed in.
- `return` sends a value back to the caller and ends the function at once.

A regular function that reaches its end without returning a value gives back `undefined`. So **return** the answer, and let the caller decide whether to print it:

```
function area(width, height) {
  return width * height;
}

const a = area(4, 3);
console.log(a + 1);          // 13  (a returned value can be used again)
```

Arrow functions

An arrow function is a shorter way to write a function. Store it in a `const`:

```
const square = (n) => n * n;
const add = (a, b) => a + b;

console.log(square(5));      // 25
console.log(add(2, 3));      // 5
```

- With one expression after `=>`, its value is returned without writing `return`.
- With curly braces you need `return` again:

```
const grade = (mark) => {
  if (mark >= 50) {
    return "pass";
  }
  return "fail";
};
console.log(grade(64));      // pass
```

Arrow functions are often passed to other functions, as you will see with `map` and `filter` in topic 7.

Scope

The scope of a variable is the part of the code that can see it. A variable made with `let` or `const` inside `{ }` is accessible in that block and its nested scopes. A nested function can keep access after the outer call returns:

```
const level = "global";

function show() {
  const inside = "local";
  console.log(level);        // global  (outer variables are visible)
  console.log(inside);       // local
}

show();
console.log(typeof inside);  // undefined  (inside does not exist out
↪  here)
```

- A variable made inside a function is a local variable ; each call gets a fresh one.
- Pass values **in** as parameters and get results **out** with `return`, instead of changing outer variables from inside a function.

Common mistakes

- Forgetting `return`: the function works out the answer, then gives back `undefined`.
- Calling without brackets: `greet` is the function itself; `greet("Mei")` runs it.
- Printing inside the function **and** printing its result prints twice, or prints `undefined`.