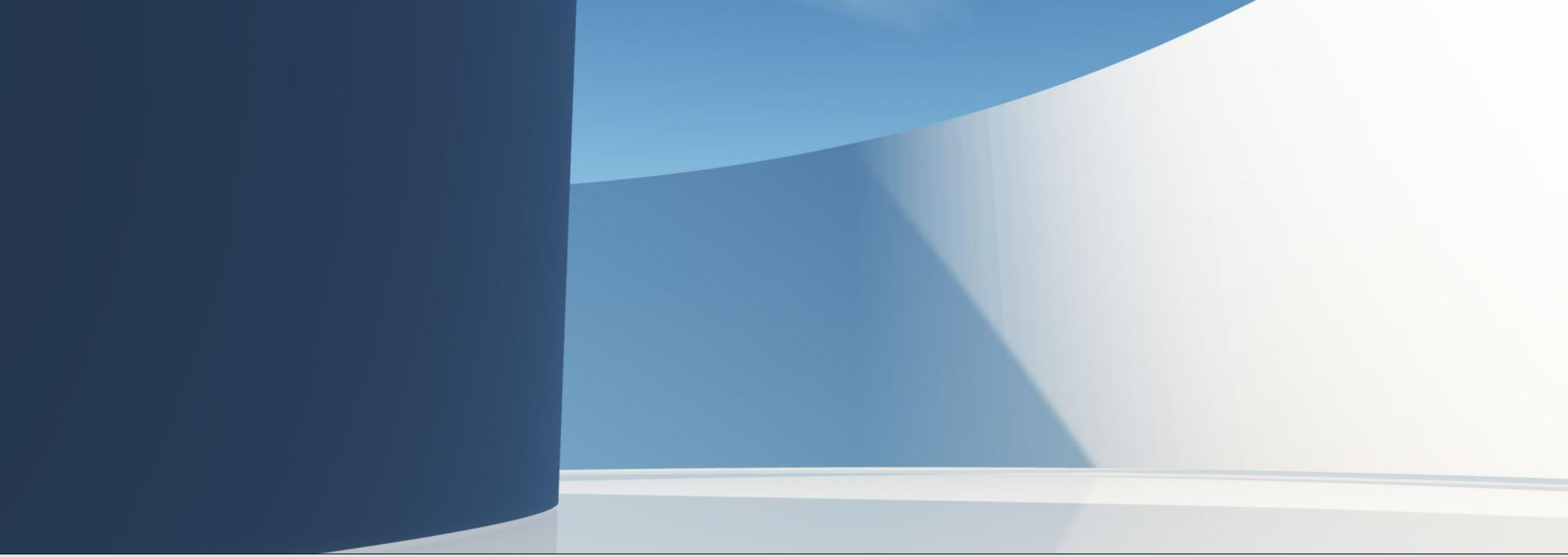


ARABIC TO ROMAN NUMERAL

Comparing Algorithms

JUST TO REPEAT...

- Simple Algorithm
- 1 = I, 5 = V but 4 is IV
- X = 10 but 9 is IX
- L = 50 but 40 = XL
- 100 = C but 90 = XC
- We stop here :)

A FIRST APPROACH

```
function roman(number) {
  let result = "";
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  if (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  if (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  if (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  if (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

Evaluation

- Very verbose
- Many (visual) repetitions
- Simple to understand – we will get back to this!
- It actually works

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  if (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  if (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  if (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  if (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

Progress

- Making similar things the same

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  while (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  while (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  while (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  while (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

Progress

- Making similar things the same

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"},
    {arabic:90, roman:"XC"},
    {arabic:50, roman:"L"},
    {arabic:40, roman:"XL"},
    {arabic:10, roman:"X"},
    {arabic:9, roman:"IX"},
    {arabic:5, roman:"V"},
    {arabic:4, roman:"IV"},
    {arabic:1, roman:"I"},
  ];
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  while (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  while (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  while (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  while (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

Progress

- Making similar things the same
- Defining the pattern map

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  while (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  while (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  while (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  while (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

Data Drives your Code!

Progress

- Making similar things the same
- Defining the pattern map, and align it

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion in map) {
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
  }
  return result;
}
```

Progress

- Making similar things the same
- Defining the pattern map, and align it
- Replace all cases with one loop

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
  }
  return result;
}
```

Progress

- Making similar things the same
- Defining the pattern map, and align it
- Replace all cases with one loop
- Replace the “for in” loop with “for of”, because you will always pick the wrong one the first time around.

WE CAN DO BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
  }
  return result;
}
```

Evaulation

- Less verbose
  - *Objectively shorter*
- Less visual repetitions
  - *I would argue, it has only the required visual repetitions*
- It still works
- Is it still simple to understand?

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
  }
  return result;
}
```

Progress

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    /*
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
    */
  }
  return result;
}
```

Progress

- We can get rid of the inner loop

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    result += conversion.roman.repeat(number / conversion.arabic);
    /*
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
    */
  }
  return result;
}
```

Progress

- We can get rid of the inner loop
- We append roman numerals repeatedly, so just do that – how many times? “As many as fit” = division

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    result += conversion.roman.repeat(number / conversion.arabic);
    number %= conversion.arabic;
    /*
    while (number >= conversion.arabic) {
      result += conversion.roman;
      number -= conversion.arabic;
    }
    */
  }
  return result;
}
```

Progress

- We can get rid of the inner loop
- We append roman numerals repeatedly, so just do that – how many times? “As many as fit” = division
- We reduce the number on each step by “as many as fit” but keep the rest = modulo

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    result += conversion.roman.repeat(number / conversion.arabic);
    number %= conversion.arabic;
  }
  return result;
}
```

Progress

- We can get rid of the inner loop
- We append roman numerals repeatedly, so just do that – how many times? “As many as fit” = division
- We reduce the number on each step by “as many as fit” but keep the rest = modulo

WE CAN DO EVEN BETTER

```
function roman(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    result += conversion.roman.repeat(number / conversion.arabic);
    number %= conversion.arabic;
  }
  return result;
}
```

Evaluation

- Simplified the complexity by removing one loop…
- …by adding mathematical complexity.
- Is it still simple to understand? Is it understandable at all?
- Could you explain this algorithm to a child?

WHAT ABOUT RECURSION?

WHAT ABOUT RECURSION?

```
const map = [
  {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
  {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
  {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
  {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
  {arabic:1,   roman:"I"},
];
```

WHAT ABOUT RECURSION?

```
const map = [
  {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
  {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
  {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
  {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
  {arabic:1,   roman:"I"},
];

function roman(number, result = "") {
  if (number === 0) {
    return result;
  }
}
```

WHAT ABOUT RECURSION?

```
const map = [
  {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
  {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
  {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
  {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
  {arabic:1,   roman:"I"},
];

function roman(number, result = "") {
  if (number === 0) {
    return result;
  }
  for (const conversion of map) {
    if (number >= conversion.arabic) {
      return roman(
        number - conversion.arabic,
        result + conversion.roman
      );
    }
  }
}
```

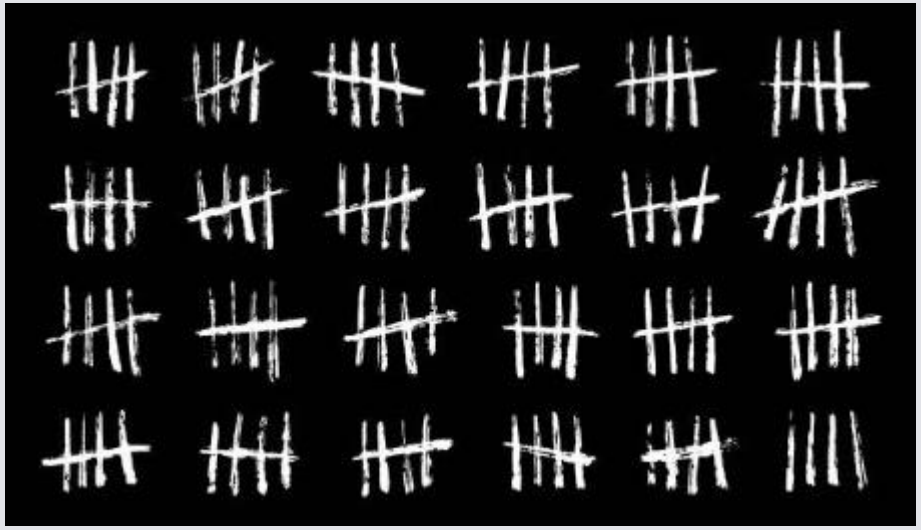
WHAT ABOUT RECURSION?

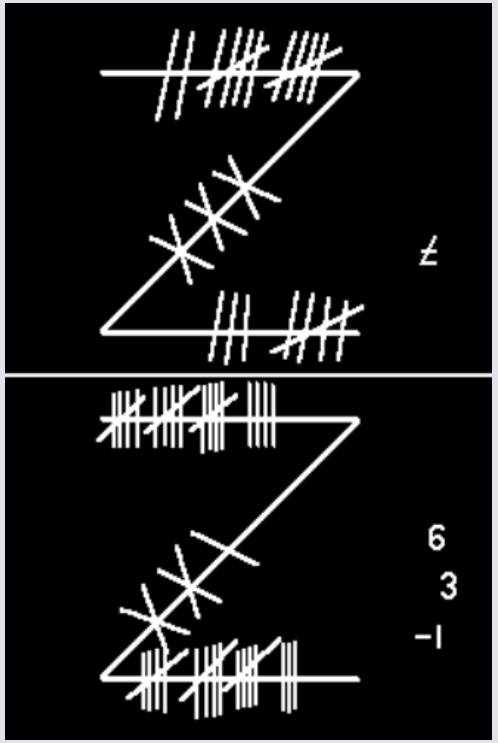
```
const map = [
  {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
  {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
  {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
  {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
  {arabic:1,   roman:"I"},
];

function roman(number, result = "") {
  if (number === 0) {
    return result;
  }
  for (const conversion of map) {
    if (number >= conversion.arabic) {
      return roman(
        number - conversion.arabic,
        result + conversion.roman
      );
    }
  }
}
```

Evaluation

- Simplified the “complex” mathematical operation for the “cost” of recursion.
- Is it still simple to understand? Is it understandable at all?
- Could you explain this algorithm to a child?

LET'S START FROM SCRATCH





LET'S START FROM SCRATCH

```
function roman(number) {
  return "";
}
```

Progress

- What is a roman number?

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number);
}

/*
roman(42);'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII'
roman(99);'IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII...'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V");
}

/*
roman(42);'VVVVVVVVII'
roman(99);'VVVVVVVVVVVVVVVVVIIII'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!
- To simplify, replace 5 consecutive strokes with a "V"

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV");
}

/*
roman(42);'VVVVVVVVII'
roman(99);'VVVVVVVVVVVVVVVVVIV'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!
- To simplify, replace 5 consecutive strokes with a "V"
  - *Special case: write "IV" instead of "IIII"*

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV")
            .replaceAll("VV",    "X");
}

/*
roman(42);'XXXXII'
roman(99);'XXXXXXXXXVIV'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!
- To simplify, replace 5 consecutive strokes with a "V"
  - *Special case: write "IV" instead of "IIII"*
- To simplify, replace 2 consecutive "V" with a "X"

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV")
            .replaceAll("VV",    "X").replace("VIV",  "IX");
}

/*
roman(42);'XXXXII'
roman(99);'XXXXXXXXXIX'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!
- To simplify, replace 5 consecutive strokes with a "V"
  - *Special case: write "IV" instead of "IIII"*
- To simplify, replace 2 consecutive "V" with a "X"
  - *Special case: write "IX" instead of "VIV"*

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV")
            .replaceAll("VV",    "X").replace("VIV",  "IX")
            .replaceAll("XXXXX", "L").replace("XXXX", "XL")
            .replaceAll("LL",    "C").replace("LXL",  "XC");
}

/*
roman(42);'XLII'
roman(99);'XCIX'
*/
```

Progress

- What is a roman number? Just the number of strokes on the wall!
- To simplify, replace 5 consecutive strokes with a "V"
  - *Special case: write "IV" instead of "IIII"*
- To simplify, replace 2 consecutive "V" with a "X"
  - *Special case: write "IX" instead of "VIV"*
- ... And so on!

LET'S START FROM SCRATCH

```
function roman(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV")
            .replaceAll("VV", "X").replace("VIV", "IX")
            .replaceAll("XXXXX", "L").replace("XXXX", "XL")
            .replaceAll("LL", "C").replace("LXL", "XC");
}
```

Evaulation

- No complex mathematical operations, no loops (hidden in repeat / replace), just simple operations
- Again simple to understand
- You could explain this algorithm to a child!

LET'S START FROM SCRATCH

```
function roman(number) {
  const map = [
    {search: "IIIII", replace: "V"}, {search: "IIII", replace: "IV"},
    {search: "VV",    replace: "X"}, {search: "VIV",  replace: "IX"},
    {search: "XXXXX", replace: "L"}, {search: "XXXX", replace: "XL"},
    {search: "LL",    replace: "C"}, {search: "LXL",  replace: "XC"},
  ];
  let result = "I".repeat(number);
  for (const conversion of map) {
    result = result.replaceAll(conversion.search, conversion.replace);
  }
  return result;
}
```

Evaulation

- No complex mathematical operations, no loops (hidden in repeat / replace), just simple operations
- Again simple to understand
- You could explain this algorithm to a child!
- You could continue to refactor...

LET'S START FROM SCRATCH

```
function roman(number) {
  const map = [
    {search: "IIIII", replace: "V"}, {search: "IIII", replace: "IV"},
    {search: "VV",    replace: "X"}, {search: "VIV",  replace: "IX"},
    {search: "XXXXX", replace: "L"}, {search: "XXXX", replace: "XL"},
    {search: "LL",    replace: "C"}, {search: "LXL",  replace: "XC"},
  ];
  const init = "I".repeat(number);
  return map.reduce(function (result, conversion) {
    return result.replaceAll(conversion.search, conversion.replace);
  }, init);
}
```

Evaulation

- No complex mathematical operations, no loops (hidden in repeat / replace), just simple operations
- Again simple to understand
- You could explain this algorithm to a child!
- You could continue to refactor...

COMPARISON
WHICH SOLUTION WOULD YOU PREFER?

```
function roman(number) {
  let result = "";
  while (number >= 100) {
    result += "C";
    number -= 100;
  }
  if (number >= 90) {
    result += "XC";
    number -= 90;
  }
  while (number >= 50) {
    result += "L";
    number -= 50;
  }
  if (number >= 40) {
    result += "XL";
    number -= 40;
  }
  while (number >= 10) {
    result += "X";
    number -= 10;
  }
  if (number >= 9) {
    result += "IX";
    number -= 9;
  }
  while (number >= 5) {
    result += "V";
    number -= 5;
  }
  if (number >= 4) {
    result += "IV";
    number -= 4;
  }
  while (number >= 1) {
    result += "I";
    number -= 1;
  }
  return result;
}
```

```
function romanBetter(number) {
  let result = "";
  const map = [
    {arabic:100, roman:"C"}, {arabic:90, roman:"XC"},
    {arabic:50,  roman:"L"}, {arabic:40, roman:"XL"},
    {arabic:10,  roman:"X"}, {arabic:9,  roman:"IX"},
    {arabic:5,   roman:"V"}, {arabic:4,  roman:"IV"},
    {arabic:1,   roman:"I"},
  ];
  for (const conversion of map) {
    result += conversion.roman.repeat(number / conversion.arabic);
    number %= conversion.arabic;
  }
  return result;
}
```

```
function romanSimple(number) {
  return "I".repeat(number)
            .replaceAll("IIIII", "V").replace("IIII", "IV")
            .replaceAll("VV",    "X").replace("VIV",  "IX")
            .replaceAll("XXXXX", "L").replace("XXXX", "XL")
            .replaceAll("LL",    "C").replace("LXL",  "XC");
}
```

```
function romanSimpleAndMaybeBetter(number) {
  const map = [
    {search: "IIIII", replace: "V"}, {search: "IIII", replace: "IV"},
    {search: "VV",    replace: "X"}, {search: "VIV",  replace: "IX"},
    {search: "XXXXX", replace: "L"}, {search: "XXXX", replace: "XL"},
    {search: "LL",    replace: "C"}, {search: "LXL",  replace: "XC"},
  ];
  const init = "I".repeat(number);
  return map.reduce(function (result, conversion) {
    return result.replaceAll(conversion.search, conversion.replace);
  }, init);
}
```

SUMMARY

- All algorithms work and result in the same behaviour.
- It is not trivial to design elegant and easy-to-undertand and optimal and extendable and refactorable and testable and … algorithms.
  - *Spikes from XP can help*
  - *And a lot of practice – the patterns *will* repeat!*
- Try to explain your algorithm “to a child”
  - *…in your head*
  - *…or use the most-childish colleague as substitute.*
- Sometimes you will need advanced concepts, sometimes there is an easier way.

ACKNOWLEDGEMENT

Shamelessly stolen and copied from **StackOverflow** and multiple talks from **Kevlin Henney**, just google him, he's worth your time.

"All problems in computer science can be solved by another level of indirection."
- David Wheeler

"...except for the problem of too many layers of indirection."
- Kevlin Henney

David Thalmann – david.thalmann@css.ch // github.com/boast