<codeware/>

how to present code

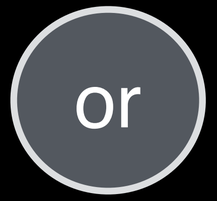
your slides are not your IDE

$> not your console either

here are five rules for
presenting code

```
rule 1:
$ use monospaced font
$
```

monospace



proportional

monospace	fixed or variable
proportional	fixed or variable

monospace

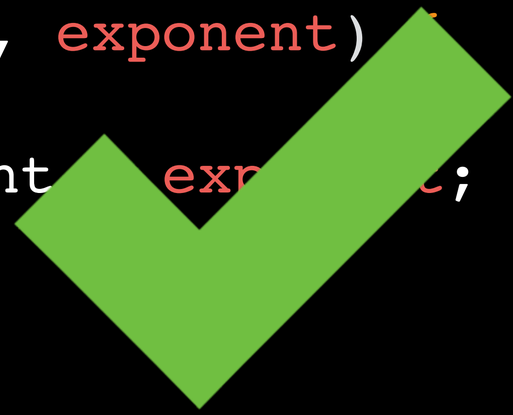
```
var power = function(base, exponent) {
  var result = 1;
  for (var count = 0; count < exponent; count++)
    result *= base;
  return result;
};
```

proportional

```
var power = function(base, exponent) {
 var result = 1;
 for (var count = 0; count < exponent; count++)
   result *= base;
 return result;
};
```

monospace

```
var power = function(base, exponent) {
  var result = 1;
  for (var count = 0; count < exponent; count++)
    result *= base;
  return result;
};
```



proportional

```
var power = function(base, exponent) {
 var result = 1;
 for (var count = 0; count < exponent; count++
   result *= base;
 return result;
};
```

monospace

proportional

monospaced fonts
have better readability
when presenting code

monospace

```
monospaced fonts
have better readability
when presenting code
```

proportional

it is harder to follow code
with proportional fonts

use monospaced fonts
for code readability

```
rule 2:
$ use big fonts
$
```

presenting for a big audience?

$ this is not your desktop monitor
$ font size should be bigger
$

I mean BIGGER
than your think

I really mean BIGGER
than your think

```
console.log(“even BIGGER where possible”)
```

```
console
   .log(“use line breaks”)
```

```
<body>
   <button onClick=“coolFunction()” autofocus>too small</button>
</body>
```

```
<body>
    <button
        onClick="coolFunction()"
        autofocus>
        same but BIGGER
    </button>
</body>
```

BIGGER is better

how do you expect people in the back to read this?

```
rule 3:
$ smart syntax
  highlighting
$
```

this is not your IDE
use syntax highlighting
only where needed

this is not your IDE
use syntax highlighting
only where needed

Presenting code on your slides usually results in a wall of text. Highlight only the specific parts you want your audience to focus on.

Presenting code on your slides usually results in a wall of text. Highlight only the specific parts you want your audience to focus on.

Presenting code on your slides usually results in a wall of text. Highlight only the specific parts you want your audience to focus on.

3 slides example - Ruby map function

3 slides example - Ruby map function

```ruby
names = ['rey', 'fin', 'poe']

names.map! {|name| name.capitalize }

puts names
```

3 slides example - Ruby map function

```
names = ['rey', 'fin', 'poe']

names.map! {|name| name.capitalize }

puts names
```

3 slides example - Ruby map function

```ruby
names = ['rey', 'fin', 'poe']

names.map! {|name| name.capitalize }

puts names
```

```
# output
Rey
Fin
Poe
```

rule 4:

```
$ using ellipsis…
$ 
```

Solving the N-queens Problem

```
import sys

from ortools.constraint_solver import pywrapcp

# By default, solve the 8x8 problem.
n = 8 if len(sys.argv) < 2 else int(sys.argv[1])

# Creates the solver.
solver = pywrapcp.Solver("n-queens")
# Creates the variables.
# The array index is the row, and the value is the column.
queens = [solver.IntVar(0, n - 1, "x%i" % i) for i in range(n)]
# Creates the constraints.

# All columns must be different.
# (The "not on the same row" constraint is implicit from the array we're
# using to store our variables; since no two elements of an array
# can have the same index, there can be no queens with the same row.)
solver.Add(solver.AllDifferent(queens))

# No two queens can be on the same diagonal.
solver.Add(solver.AllDifferent([queens[i] + i for i in range(n)]))
solver.Add(solver.AllDifferent([queens[i] - i for i in range(n)]))
```

```
db = solver.Phase(queens,
                  solver.CHOOSE_MIN_SIZE_LOWEST_MAX,
                  solver.ASSIGN_CENTER_VALUE)

solver.NewSearch(db)

# Iterates through the solutions, displaying each.
num_solutions = 0
while solver.NextSolution():
  queen_columns = [int(queens[i].Value()) for i in range(n)]

  # Displays the solution just computed.
  for i in range(n):
    for j in range(n):
      if queen_columns[i] == j:
        print "Q",
      else:
        print "_",
    print
  print
  num_solutions += 1

solver.EndSearch()

print
print "Solutions found:", num_solutions
print "Time:", solver.WallTime(), "ms"
```

Solving the N-queens Problem

```
import sys

from ortools.constraint_solver import pywrapcp

# By default, solve the 8x8 problem.
n = 8 if len(sys.argv) < 2 else int(sys.argv[1])

# Creates the solver.
solver = py
# Creates
# The array
queens = [
# Creates

# All colu
# (The "no
# using to
# can have
solver.Add

# No two queens can be on the same diagonal.
solver.Add(solver.AllDifferent([queens[i] + i for i in range(n)]))
solver.Add(solver.AllDifferent([queens[i] - i for i in range(n)]))
```

```
db = solver.Phase(queens,
                  solver.CHOOSE_MIN_SIZE_LOWEST_MAX,
                  solver.ASSIGN_CENTER_VALUE)

solver.NewSearch(db)

# Iterates through the solutions, displaying each.
num_solutions = 0

                                                    range(n)]

  print
  num_solutions += 1

solver.EndSearch()

print
print "Solutions found:", num_solutions
print "Time:", solver.WallTime(), "ms"
```

what’s the point in presenting all you code if people can’t follow?

Solving the N-queens Problem

```
import sys

from ortools.constraint_solver import pywrapcp

# By default, solve the 8x8 problem.
n = 8 if len(sys.argv) < 2 else int(sys.argv[1])

# Creates the solver.
solver = py
# Creates
# The array
queens = [
# Creates

# All colu
# (The "no
# using to
# can have
solver.Add

# No two queens can be on the same diagonal.
solver.Add(solver.AllDifferent([queens[i] + i for i in range(n)]))
solver.Add(solver.AllDifferent([queens[i] - i for i in range(n)]))
```

keep just the relevant code

```
db = solver.Phase(queens,
                  solver.CHOOSE_MIN_SIZE_LOWEST_MAX,
                  solver.ASSIGN_CENTER_VALUE)

solver.NewSearch(db)

# Iterates through the solutions, displaying each.
num_solutions = 0

                                                    range(n)]

  print
  num_solutions += 1

solver.EndSearch()

print
print "Solutions found:", num_solutions
print "Time:", solver.WallTime(), "ms"
```

Solving the N-queens Problem

```
import sys

from ortools.constraint_solver import pywrapcp

# By default, solve the 8x8 problem.
n = 8 if len(sys.argv) < 2 else int(sys.argv[1])

# Creates the solver.
solver = py
# Creates
# The array
queens = [
# Creates

# All colu
# (The "no
# using to
# can have
solver.Add

# No two queens can be on the same diagonal.
solver.Add(solver.AllDifferent([queens[i] + i for i in range(n)]))
solver.Add(solver.AllDifferent([queens[i] - i for i in range(n)]))
```

use ellipsis…

```
db = solver.Phase(queens,
                  solver.CHOOSE_MIN_SIZE_LOWEST_MAX,
                  solver.ASSIGN_CENTER_VALUE)

solver.NewSearch(db)

# Iterates through the solutions, displaying each.
num_solutions = 0

                                                    range(n)]

  print
  num_solutions += 1

solver.EndSearch()

print
print "Solutions found:", num_solutions
print "Time:", solver.WallTime(), "ms"
```

Solving the N-queens Problem

```
...
# Creates the solver.
solver = pywrapcp.Solver("n-queens")
...
# Iterates through the solutions, displaying each.
num_solutions = 0
while solver.NextSolution():
  queen_columns = [int(queens[i].Value()) for i in range(n)]

  # Displays the solution just computed.
  for i in range(n):
    for j in range(n):
      if queen_columns[i] == j:
        print "Q",
      else:
        print "_",
    print
  print
  num_solutions += 1

solver.EndSearch()
...
```

Solving the N-queens Problem

```
...
# Creates the solver.
solver = pywrapcp.Solver("n-queens")
...
# Iterates through the solutions, displaying each.
num_solutions = 0
while solver.NextSolution():
  queen_columns = [int(queens[i].Value()) for i in range(n)]

  # Displays the solution just computed.
  for i in range(n):
    for j in range(n):
      if queen_columns[i] == j:
        print "Q",
      else:
        print "_",
    print
  print
  num_solutions += 1

solver.EndSearch()
...
```

the focus is now on the algorithm
not the boilerplate

```
rule 5:
$ use screen annotation
$
```

do you want to focus your audience’s attention on a specific area in the screen?

thinking of using a laser pointer?

do you expect your audience
to track this?

always create slides as if
you are the person sitting
in the last row of the
conference hall.
use screen annotations.

always create slides as if you are the person sitting in the last row of the conference hall.

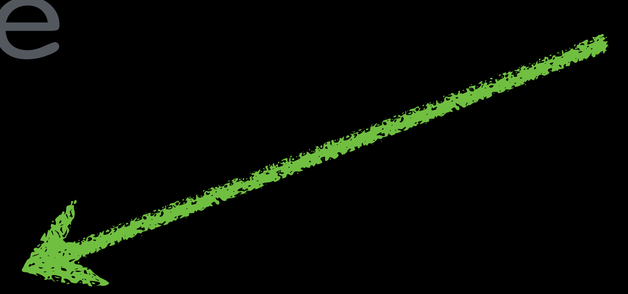
use screen annotations.

always create slides as if you are the person sitting in the last row of the conference hall.

use screen annotations.

always create slides as if you are the person sitting in the last row of the conference hall.

use screen annotations.



putting it all together

```
$ ECMAScript 2015 Promise example
$
```

putting it all together

```
$ ECMAScript 2015 Promise example
$
```

the next slides will introduce the
ECMAScript 2015 Promise Object

```
fetchGraphData(graphName, function(error, graphData)) {
  if (error) { // handle error }
  // ...
  renderGraph(graphData, function(error)) {
    if (error) { // handle error }
    // ...
    notifyClients(msg, result, function(error, response)) {
      if (error) { // handle error }
      // ...
    });
  });
});
```

Typical JS code results with many nested callbacks

```
fetchGraphData(graphName, function(error, graphData)) {
  if (error) { // handle error }
  // ...
  renderGraph(graphData, function(error)) {
    if (error) { // handle error }
    // ...
    notifyClients(msg, result, function(error, response)) {
      if (error) { // handle error }
      // ...
    });
  });
});
```

makes it hard to follow

```
fetchGraphData(graphName, function(error, graphData)) {
  if (error) { // handle error }
  // ...
  renderGraph(graphData, function(error)) {
    if (error) { // handle error }
    // ...
    notifyClients(msg, result, function(error, response)) {
      if (error) { // handle error }
      // ...
    });
  });
});
```

error handling is duplicated

ECMAScript 2015 Promise
to the rescue!

```
function fetchGraphData(graphName) {
  return new Promise(function(resolve, reject)) {
    let request = new XMLHttpRequest();
    ...

    // XMLHttpRequest to server
    request.open('GET', 'http://example.com', true);
    request.onload = function() {
      if (request.status == 200) {
        resolve(request.response);
      }
      else {
        reject(new Error(request.status));
      }
    };
});
```

The Promise object is used for deferred and asynchronous computations

```
function fetchGraphData(graphName) {
  return new Promise(function(resolve, reject)) {
    let request = new XMLHttpRequest();
    ...

    // XMLHttpRequest to server
    request.open('GET', 'http://example.com', true);
    request.onload = function() {
      if (request.status == 200) {
        resolve(request.response);
      }
      else {
        reject(new Error(request.status));
      }
    };
});
```

resolve is called once operation has completed successfully

```
function fetchGraphData(graphName) {
  return new Promise(function(resolve, reject)) {
    let request = new XMLHttpRequest();
    ...

    // XMLHttpRequest to server
    request.open('GET', 'http://example.com', true);
    request.onload = function() {
      if (request.status == 200) {
        resolve(request.response);
      }
      else {
        reject(new Error(request.status));
      }
    };
});
```

reject is called if the operation has been rejected

Calling a Promise

```
fetchGraphData(graphName)
  .then(renderGraph)
  .then(notifyClients)
  .catch(function(error)) {
    console.log(“Error:”, error)
  });
```

Promise code is easy to read

Calling a Promise

```
fetchGraphData(graphName)
  .then(renderGraph)
  .then(notifyClients)
  .catch(function(error)) {
    console.log(“Error:”, error)
  });
```

you can nest multiple Promise calls

Calling a Promise

```
fetchGraphData(graphName)
  .then(renderGraph)
  .then(notifyClients)
  .catch(function(error)) {
    console.log(“Error:”, error)
  });
```

one error handling call

remember!

your slides are not your IDE

codeware - 5 rules

1

2

3

4

5

codeware - 5 rules

monospaced font

2

3

4

5

codeware - 5 rules

monospaced font

BIG

3

4

5

codeware - 5 rules

monospaced font

BIG

smart syntax highlighting

4

5

codeware - 5 rules

monospaced font

BIG

smart syntax highlighting

ellipsis...

5

codeware - 5 rules

monospaced font

BIG

smart syntax highlighting

ellipsis...

screen annotation

Credits

The Noun Project:
  Check by useiconic.com
  Close by Thomas Drach

N-queens problem:
  OR-tools solution to the N-queens problem.
  Inspired by Hakan Kjellerstrand's solution at
  http://www.hakank.org/google_or_tools/,
  which is licensed under the Apache License, Version 2.0:
  http://www.apache.org/licenses/LICENSE-2.0

  https://developers.google.com/optimization/puzzles/queens#python

for more tips
follow me on twitter and slideshare

lookatmyslides