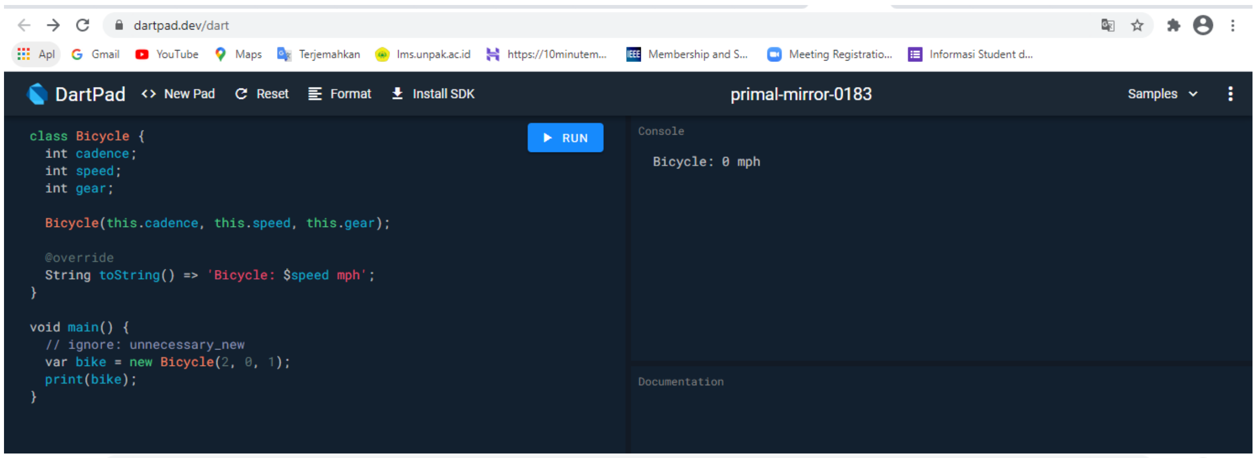
TUGAS 1 – MOBILE PROGRAMMING

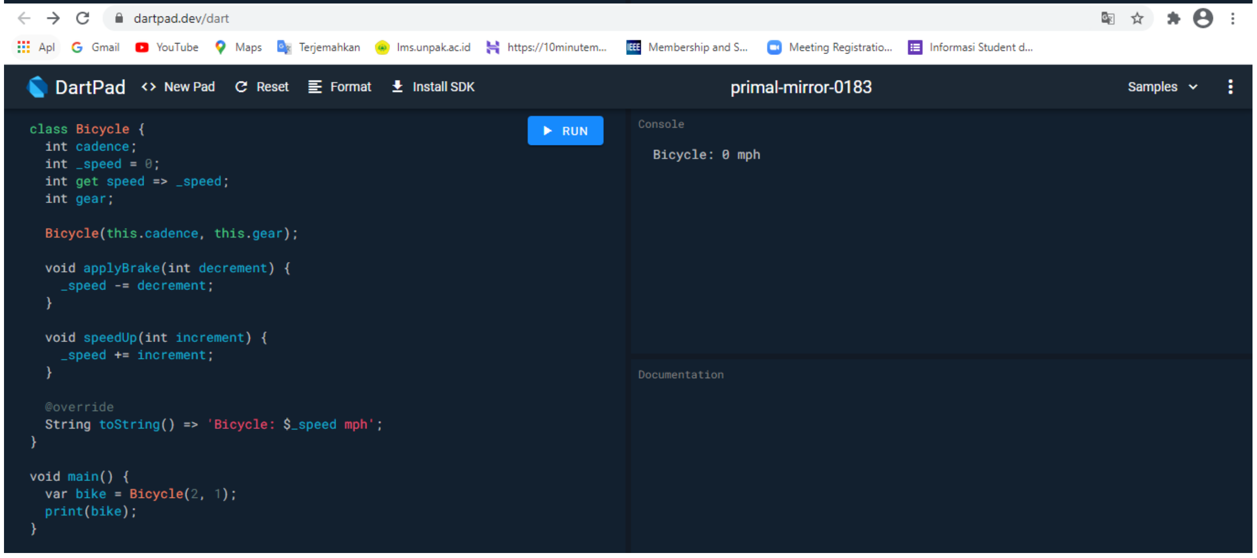
MARIA QIBTIA – 065118230

1. Create a simple Dart class – Bicycle.dart



```
class Bicycle {
  int cadence;
  int speed;
  int gear;

  Bicycle(this.cadence, this.speed, this.gear);

  @override
  String toString() => 'Bicycle: $speed mph';
}

void main() {
  // ignore: unnecessary_new
  var bike = new Bicycle(2, 0, 1);
  print(bike);
}
```

```
Bicycle: 0 mph
```

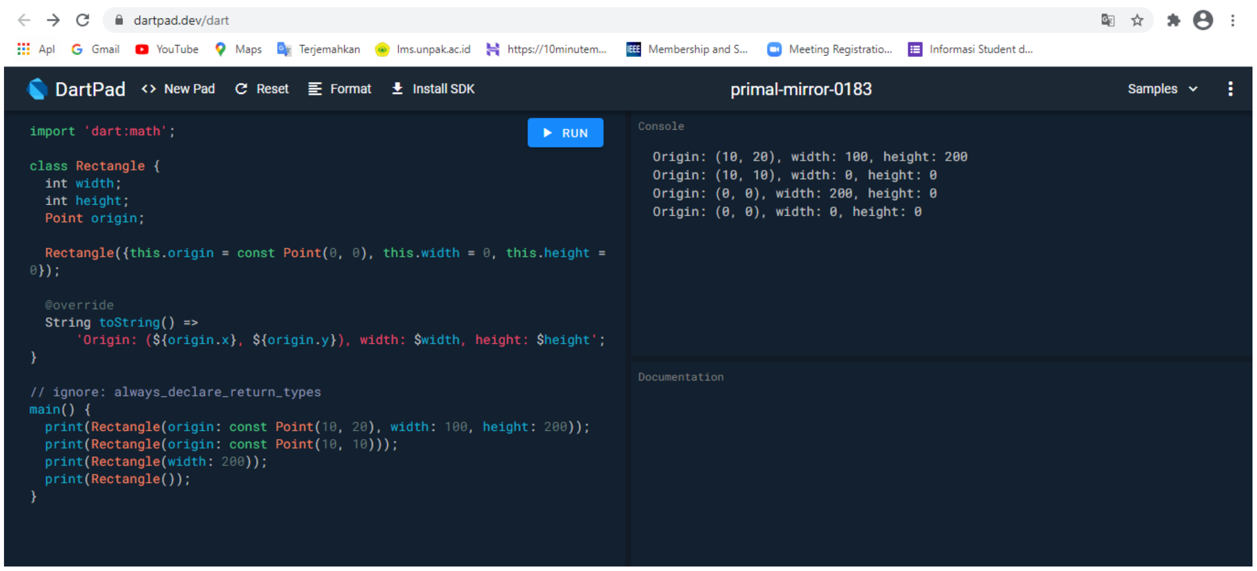


```
class Bicycle {
  int cadence;
  int _speed = 0;
  int get speed => _speed;
  int gear;

  Bicycle(this.cadence, this.gear);

  void applyBrake(int decrement) {
    _speed -= decrement;
  }

  void speedUp(int increment) {
    _speed += increment;
  }

  @override
  String toString() => 'Bicycle: $_speed mph';
}

void main() {
  var bike = Bicycle(2, 1);
  print(bike);
}
```

```
Bicycle: 0 mph
```

```
class Bicycle {
  int cadence;
  int _speed = 0;
  int get speed => _speed;
  int gear;

  Bicycle(this.cadence, this.gear);

  void applyBrake(int decrement) {
    _speed -= decrement; }
  void speedUp(int increment) {
    _speed += increment; }

  @override
  String toString() => 'Bicycle: $_speed
mph'; }

void main() {
  var bike = Bicycle(2, 1);
  print(bike); }
```

2. Gunakan parameter opsional (bukan overloading) - Rectangle.dart



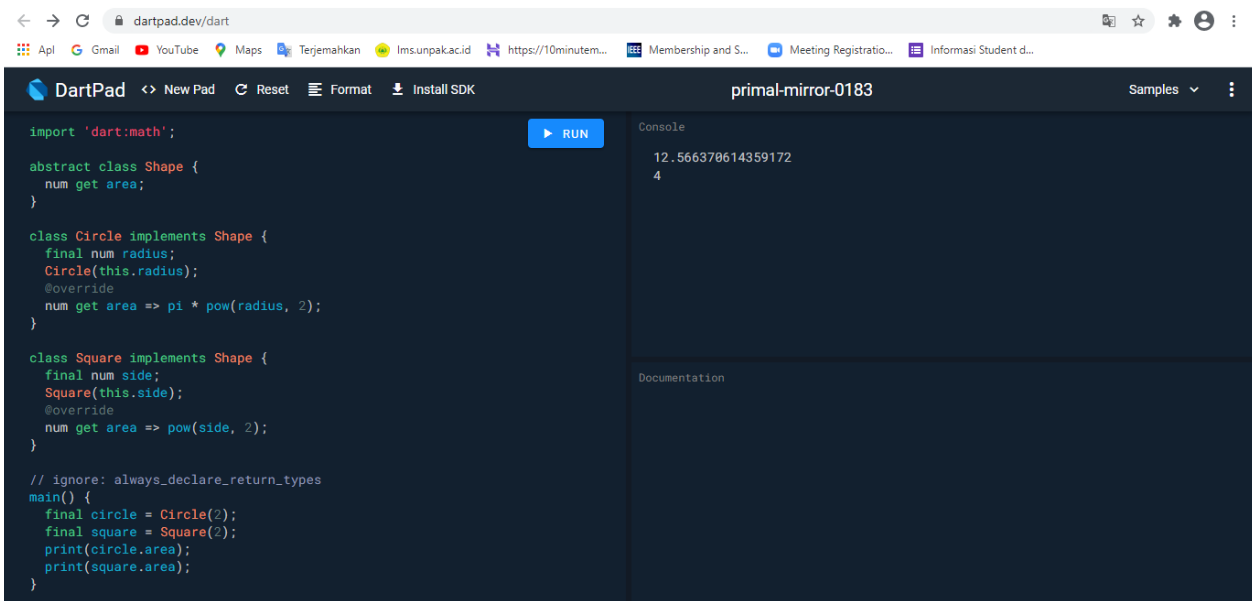
```
import 'dart:math';

class Rectangle {
  int width;
  int height;
  Point origin;

  Rectangle({this.origin = const Point(0, 0), this.width = 0, this.height =
0});

  @override
  String toString() =>
      'Origin: (${origin.x}, ${origin.y}), width: $width, height: $height';
}

// ignore: always_declare_return_types
main() {
  print(Rectangle(origin: const Point(10, 20), width: 100, height: 200));
  print(Rectangle(origin: const Point(10, 10)));
  print(Rectangle(width: 200));
  print(Rectangle());
}
```

```
Origin: (10, 20), width: 100, height: 200
Origin: (10, 10), width: 0, height: 0
Origin: (0, 0), width: 200, height: 0
Origin: (0, 0), width: 0, height: 0
```

```
import 'dart:math';

class Rectangle {
  int width;
  int height;
  Point origin;

  Rectangle({this.origin = const Point(0, 0), this.width = 0, this.height = 0});

  @override
  String toString() =>
      'Origin: (${origin.x}, ${origin.y}), width: $width, height: $height'; }

// ignore: always_declare_return_types
main() {
  print(Rectangle(origin: const Point(10, 20), width: 100, height: 200));
  print(Rectangle(origin: const Point(10, 10)));
  print(Rectangle(width: 200));
  print(Rectangle());}
```

3. **Create a factory**

Shape.dart

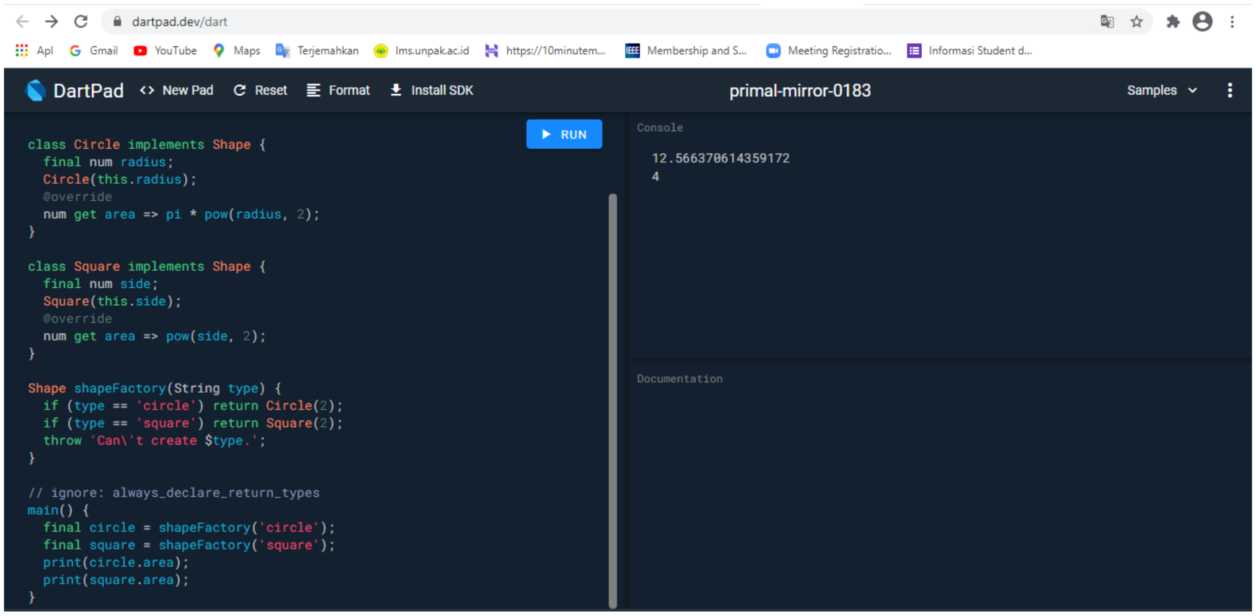


```
import 'dart:math';

abstract class Shape {
  num get area;
}

class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  @override
  num get area => pi * pow(radius, 2);
}

class Square implements Shape {
  final num side;
  Square(this.side);
  @override
  num get area => pow(side, 2);
}

// ignore: always_declare_return_types
main() {
  final circle = Circle(2);
  final square = Square(2);
  print(circle.area);
  print(square.area);
}
```

```
12.566370614359172
4
```

```
import 'dart:math';

abstract class Shape {
  num get area; }

class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  num get area => pi * pow(radius, 2);
}

class Square implements Shape {
  final num side;
  Square(this.side);
  num get area => pow(side, 2); }

main() {
  final circle = Circle(2);
  final square = Square(2);
  print(circle.area);
  print(square.area);
}
```

Top-level.dart

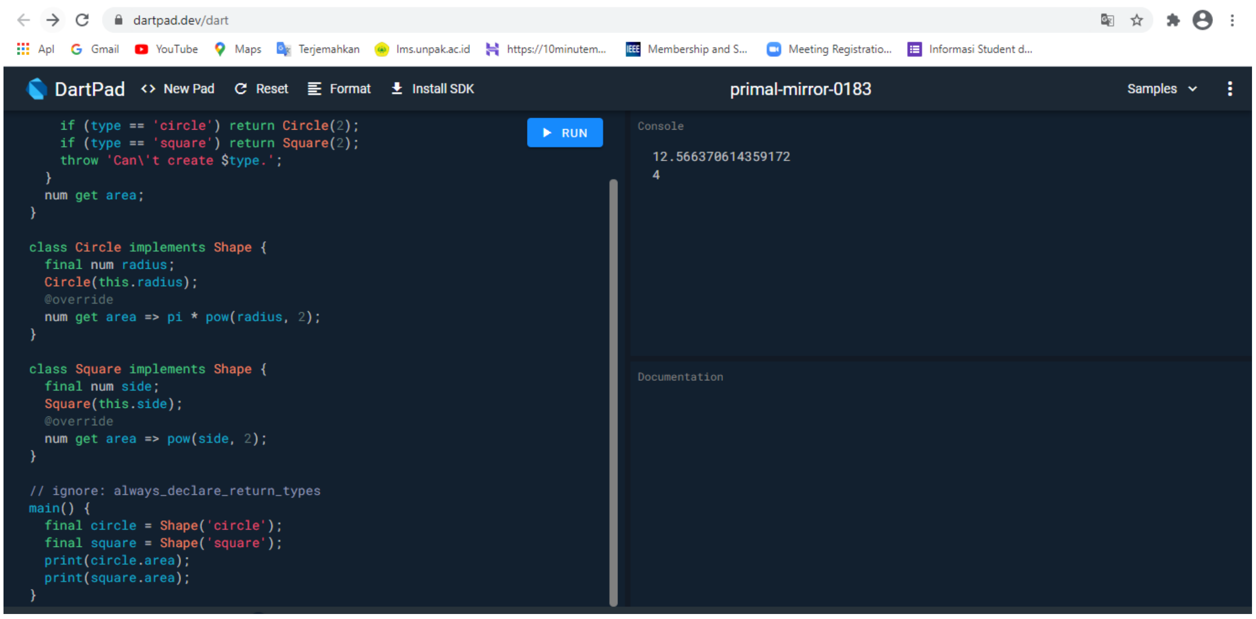


```dart
class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  @override
  num get area => pi * pow(radius, 2);
}

class Square implements Shape {
  final num side;
  Square(this.side);
  @override
  num get area => pow(side, 2);
}

Shape shapeFactory(String type) {
  if (type == 'circle') return Circle(2);
  if (type == 'square') return Square(2);
  throw 'Can\'t create $type.';
}

// ignore: always_declare_return_types
main() {
  final circle = shapeFactory('circle');
  final square = shapeFactory('square');
  print(circle.area);
  print(square.area);
}
```

```
12.566370614359172
4
```

```dart
import 'dart:math';


abstract class Shape {
  num get area; }


class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  num get area => pi * pow(radius, 2); }


class Square implements Shape {
  final num side;
  Square(this.side);
  num get area => pow(side, 2); }


Shape shapeFactory(String type) {
  if (type == 'circle') return Circle(2);
  if (type == 'square') return Square(2);
  throw 'Can\'t create $type.'; }


main() {
  final circle = shapeFactory('circle');
  final square = shapeFactory('square');
  print(circle.area);
  print(square.area); }
```

FactoryConstructor.dart

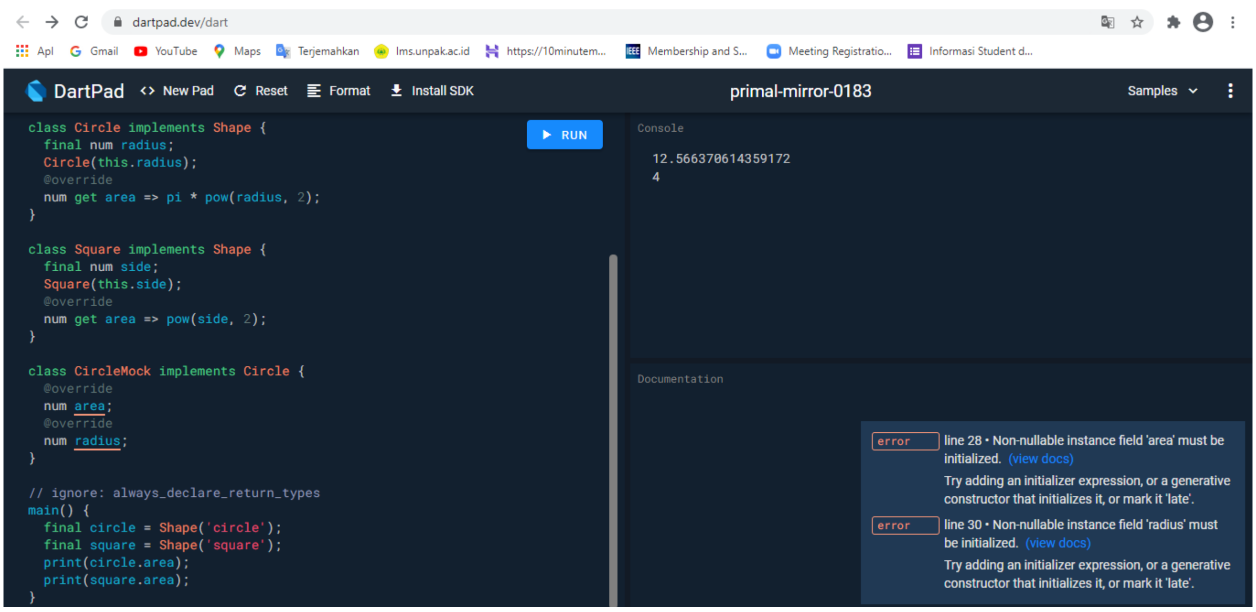


```
    if (type == 'circle') return Circle(2);
    if (type == 'square') return Square(2);
    throw 'Can\'t create $type.';
  }
  num get area;
}

class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  @override
  num get area => pi * pow(radius, 2);
}

class Square implements Shape {
  final num side;
  Square(this.side);
  @override
  num get area => pow(side, 2);
}

// ignore: always_declare_return_types
main() {
  final circle = Shape('circle');
  final square = Shape('square');
  print(circle.area);
  print(square.area);
}
```

```
12.566370614359172
4
```

```
import 'dart:math';

abstract class Shape {
  factory Shape(String type) {
    if (type == 'circle') return Circle(2);
    if (type == 'square') return Square(2);
    throw 'Can\'t create $type.'; }
  num get area; }

class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  num get area => pi * pow(radius, 2); }

class Square implements Shape {
  final num side;
  Square(this.side);
  num get area => pow(side, 2); }

main() {
  final circle = Shape('circle');
  final square = Shape('square');
  print(circle.area);
  print(square.area);  }
```

4. Implement an interface – CircleMock.dart



```dart
class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  @override
  num get area => pi * pow(radius, 2);
}

class Square implements Shape {
  final num side;
  Square(this.side);
  @override
  num get area => pow(side, 2);
}

class CircleMock implements Circle {
  @override
  num area;
  @override
  num radius;
}

// ignore: always_declare_return_types
main() {
  final circle = Shape('circle');
  final square = Shape('square');
  print(circle.area);
  print(square.area);
}
```

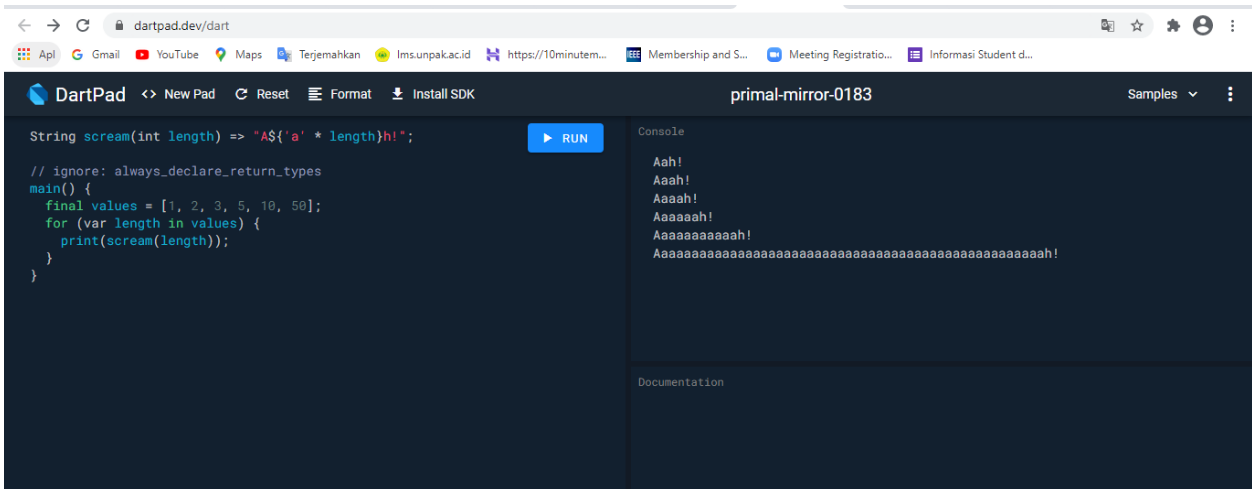
```
12.566370614359172
4
```

```
error  line 28 • Non-nullable instance field 'area' must be initialized. (view docs)
       Try adding an initializer expression, or a generative constructor that initializes it, or mark it 'late'.
error  line 30 • Non-nullable instance field 'radius' must be initialized. (view docs)
       Try adding an initializer expression, or a generative constructor that initializes it, or mark it 'late'.
```

```
import 'dart:math';


abstract class Shape {
  factory Shape(String type) {
    if (type == 'circle') return Circle(2);
    if (type == 'square') return Square(2);
    throw 'Can\'t create $type.'; }
  num get area; }


class Circle implements Shape {
  final num radius;
  Circle(this.radius);
  num get area => pi * pow(radius, 2); }


class Square implements Shape {
  final num side;
  Square(this.side);
  num get area => pow(side, 2); }


class CircleMock implements Circle {
  num area;
  num radius; }


main() {
  final circle = Shape('circle');
  final square = Shape('square');
  print(circle.area);
  print(square.area); }
```

5. **Use Dart for functional programming**

Scream1.dart

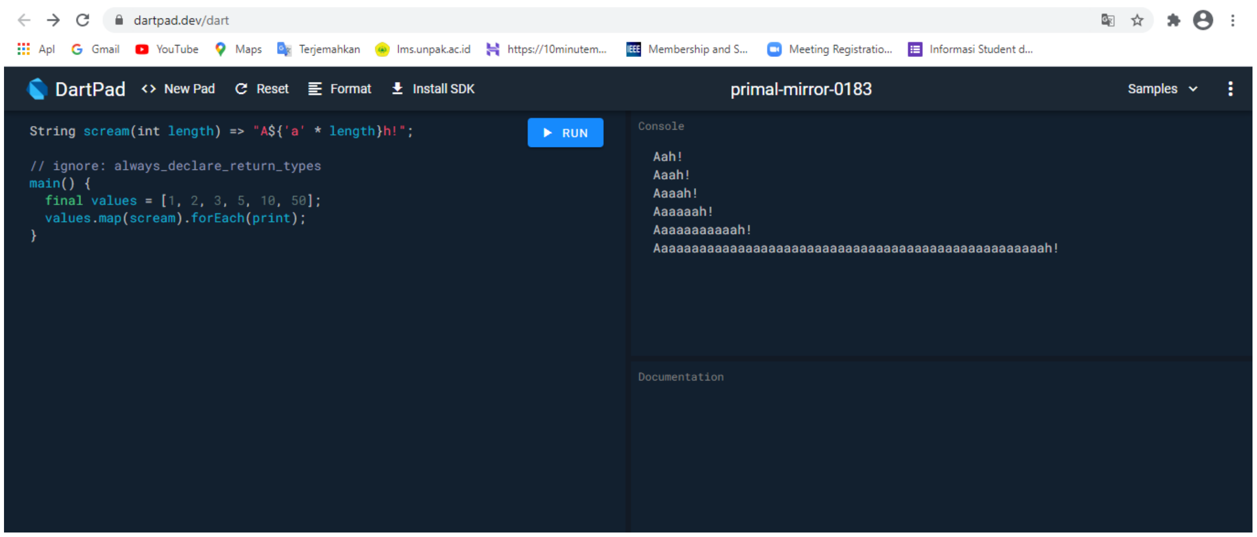


```
String scream(int length) => "A${'a' * length}h!";

// ignore: always_declare_return_types
main() {
  final values = [1, 2, 3, 5, 10, 50];
  for (var length in values) {
    print(scream(length));
  }
}
```

```
Aah!
Aaah!
Aaaah!
Aaaaaah!
Aaaaaaaaaaah!
Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaah!
```

```
String scream(int length) => "A${'a' * length}h!";


main() {
  final values = [1, 2, 3, 5, 10, 50];
  for (var length in values) {
    print(scream(length));
  }
}
```

Scream2.dart

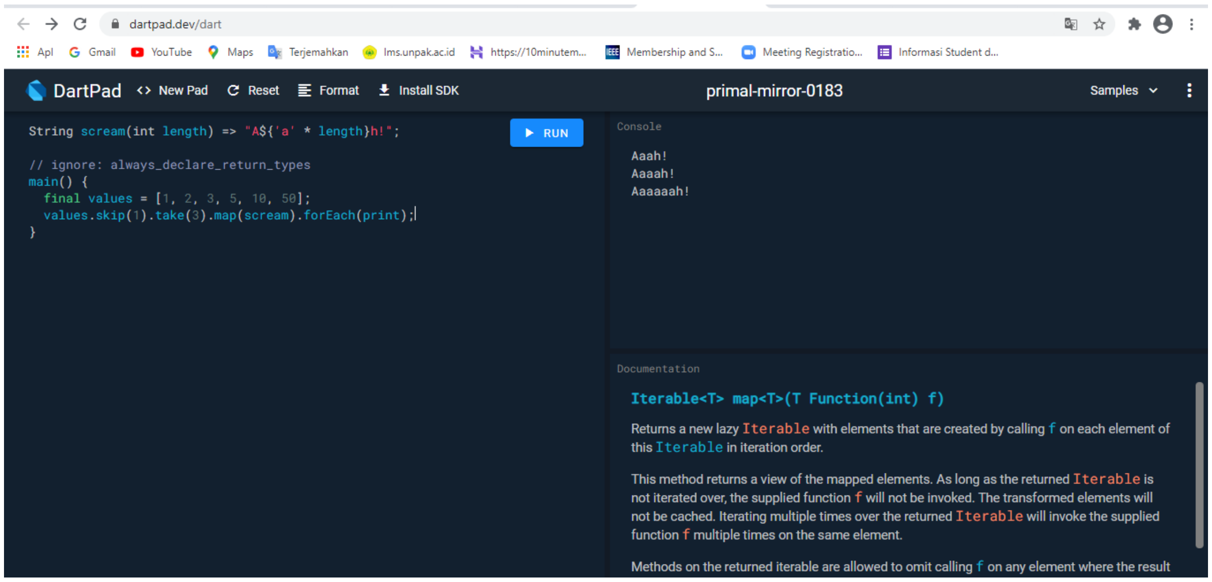


```
String scream(int length) => "A${'a' * length}h!";

// ignore: always_declare_return_types
main() {
  final values = [1, 2, 3, 5, 10, 50];
  values.map(scream).forEach(print);
}
```

```
Aah!
Aaah!
Aaaah!
Aaaaaah!
Aaaaaaaaaaah!
Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaah!
```

```
String scream(int length) => "A${'a' * length}h!";


main() {
  final values = [1, 2, 3, 5, 10, 50];
  values.map(scream).forEach(print);
}
```

Scream3.dart



```
String scream(int length) => "A${'a' * length}h!";

// ignore: always_declare_return_types
main() {
  final values = [1, 2, 3, 5, 10, 50];
  values.skip(1).take(3).map(scream).forEach(print);
}
```

```
Aaah!
Aaaah!
Aaaaaah!
```

`Iterable<T> map<T>(T Function(int) f)`

Returns a new lazy `Iterable` with elements that are created by calling `f` on each element of this `Iterable` in iteration order.

This method returns a view of the mapped elements. As long as the returned `Iterable` is not iterated over, the supplied function `f` will not be invoked. The transformed elements will not be cached. Iterating multiple times over the returned `Iterable` will invoke the supplied function `f` multiple times on the same element.

Methods on the returned iterable are allowed to omit calling `f` on any element where the result

```
String scream(int length) => "A${'a' * length}h!";


main() {
  final values = [1, 2, 3, 5, 10, 50];
  values.skip(1).take(3).map(scream).forEach(print);
}
```