# Features in Dart

Dr. Hadi Soofi

University of Tabriz

# Future

- در کد زیر، تابع Future.delayed، باعث فراخوانی تابع myPrint پس از گذشت زمان ۲ ثانیه می شود.
- یعنی تابع main()، futureCall() را فراخوانی می کند. سپس futureCall، پس از گذشت ۲ ثانیه، myPrint را فراخوانی می کند.

```
void main() {
  futureCall();
}
void futureCall() {
  Future.delayed(Duration(seconds: 2), myPrint);
}
void myPrint() {
  print("Inside function");
}
```

# Future

- هر سه بلوک کد زیر معادل هم هستند و  هیچ تفاوتی بین آن ها نیست و قسمت مشخص شده با رنگ قرمز در هر کد در واقع همان تابع فراخوانی شده پس از تکمیل Future است.

```
void futureCall() {
  Future.delayed(Duration(seconds: 2), myPrint);
}
void myPrint() {
  print("Inside function");
}
```

```
void futureCall() {
  Future.delayed(Duration(seconds: 2), () {
    print("Inside function");
  });
}
```

```
void futureCall() {
  Future.delayed(Duration(seconds: 2), () => print("inside function"));
}
```

# Future

- به کد زیر و نتیجه آن دقت کنید. همان طور که از ترتیب خروجی ها می بینید، دستور Future، روند اجرای برنامه را متوقف نکرده است. یعنی خط Third print بعد از First print چاپ شده است و سپس پس از اینکه Future تکمیل شد، خط Second print چاپ شده است.

```
void main(List<String> arguments) {
  print("First print");
  Future.delayed(Duration(seconds: 2), () => print("Second print"));
  print("Third print");
}
```

```
First print
Third print
Second print
```

# Future

- اگر بخواهید در داخل یک تابع، روند اجرا متوقف شده و پس از خاتمه Future ادامه یابد باید از کلید واژه await استفاده کنید.
- دقت کنید که این واژه را فقط قبل از توابعی می توانید استفاده کنید که یک Future برگردانند
- اگر در داخل تابعی از await استفاده کنید، آن تابع باید یک تابع async باشد (آسنکرون).
- به کد زیر و نتیجه آن دقت کنید که روند اجرا تا تکمیل Future متوقف شده و سپس Second print و بعد Third print چاپ شده است

```dart
void main(List<String> arguments) async {
  print("First print");
  await Future.delayed(Duration(seconds: 2), () => print("Second print"));
  print("Third print");
}
```

```
First print
Second print
Third print
```

# Future

- کد صفحه قبل معادل کد زیر است و تفاوتی بین آن ها نیست

```
void main(List<String> arguments) async {
  print("First print");
  await Future.delayed(Duration(seconds: 2));
  print("Second print");
  print("Third print");
}
```

# Future

- به کد زیر و نتیجه حاصل شده دقت کنید. علی رغم اینکه featurePrint() قبل از چاپ Third print فراخوانی شده است و علی رغم اینکه در داخل تابع featurePrint() از await استفاده کرده ایم، باز هم Third print زودتر چاپ شده است.
- دلیل این موضوع این است که تابع featurePrint() یک تابع async است و فراخوانی آن داخل تابع دیگری (اینجا main) روند آن تابع را مختل نمی کند

```dart
void main(List<String> arguments) async {
  print("First print");
  featurePrint();
  print("Third print");
}
Future<void> featurePrint() async {
  await Future.delayed(Duration(seconds: 2));
  print("Inside function");
}
```

```
First print
Third print
Second print
```

# Future

- در Dart و مخصوصاً Flutter، بعضی کتابخانه ها وجود دارند که به دلیل ماهیت عملکردشان یک Future برمی گردانند. مثلاً کتابخانه http که برای ارسال درخواست های http به کار می رود، از این جمله است.
- دلیل این امر آن است که مسلماً چنین درخواستی تا رسیدن به سرور و دریافت پاسخ آن طول خواهد کشید. در این حالات بایستی از روندهای گفته شده تا کنون استفاده کنیم.
- مثلاً می خواهیم توسط یک REST API رایگان، قیمت لحظه ای طلا را دریافت کنیم.
- اولین قدم نصب کتابخانه لازم توسط اجرای دستور dart pub add http در terminal است.
- از api رایگان فراهم شده توسط سایت https://www.goldapi.io/ استفاده می کنیم.
- صفحه بعد کد لازم را نشان می دهد.

# Future

- در این کد apiKey_، یک api key است که توسط این سایت به رایگان در اختیار شما قرار می گیرد.
- می بینید که http.get() به شما یک Future<Response> را برمی گرداند. بنابراین توسط await منتظر می مانیم تا درخواست تکمیل شود. حالا یک Response در اختیار داریم. توسط jsonDecode، json دریافتی را به یک Map تبدیل می کنیم و سپس مقدار کلید price را برمی گردانیم.

```
Future<double> getGoldPrice() async {
  final uri = Uri.parse('https://www.goldapi.io/api/XAU/USD');
  final response = await http.get(
    uri,
    headers: {'x-access-token': _apiKey, 'Content-type': 'application/json'},
  );
  final map = jsonDecode(response.body);
  return map['price'];
}
```

```
void main(List<String> arguments) async {
  print("Getting gold price...");
  double price = await getGoldPrice();
  print("Each ounce of gold is $price\$ now");
}
```

```
Getting gold price...
Each ounce of gold is 4039.065$ now
```

# Future

- البته لازم است وقتی با درخواست های http سر و کار داریم، statusCode پاسخ دریافتی را بررسی کنیم. چون ممکن است درخواست ما ناموفق بوده باشد.
- اینجا من به این شکل عمل کرده ام که اگر درخواست ناموفق باشد، null برگردانم. لذا نمی توانم یک Future<double> را به عنوان خروجی تعریف کنم و باید از یک nullable type استفاده کنیم یعنی double?.

```
Future<double?> getGoldPrice() async {
  final uri = Uri.parse('https://www.goldapi.io/api/XAU/USD');


  final response = await http.get(
    uri,
    headers: {'x-access-token': _apiKey, 'Content-type': 'application/json'},
  );
  if (response.statusCode == 200) {
    final map = jsonDecode(response.body);
    return map['price'];
  } else {
    return null;
  }
}
```

# Future

- اینجا هم تغییرات لازم در تابع main داده شده است تا امکان null بودن خروجی تابع پوشش داده شود.

```
void main(List<String> arguments) async {
  print("Getting gold price...");
  double? price = await getGoldPrice();
  price != null
      ? print("Each ounce of gold is $price\$ now")
      : print("Something went wrong");
}
```

# Future

- می توانید از try catch برای اطمینان بیشتر استفاده کنید.

```
Future<double?> getGoldPrice() async {
  final uri = Uri.parse('https://www.goldapi.io/api/XAU/USD');
  try {
    final response = await http.get(
      uri,
      headers: {'x-access-token': _apiKey, 'Content-type': 'application/json'},
    );
    if (response.statusCode == 200) {
      final map = jsonDecode(response.body);
      return map['price'];
    } else {
      return null;
    }
  } catch (e) {
    return null;
  }
}
```