Dart examples (second lecture about Dart)

1. Dart allows users to check for types using the ‘is’ keyword.
2. main() {
3. printType(23);
4. printType('mark');
5. }
6. printType(dynamic d) {
7. if (d is int) {
8. print('Its an Integer');
9. }
10. if (d is String) {
11. print('Its a String');
12. }
13. }

2- Dart gives the developer a way to get information about an Object’s type at runtime. You can use Object’s runtime Type property, which returns a Type object.

void main() {

  var v1 = 10;

  print(v1.runtimeType);

  var v2 = 'hello';

  print(v2.runtimeType);

}

3- Interpolation One very useful feature of Dart is its string interpolation. You can put the value of an expression inside a string by using ${expression}.

class Person {

  String firstName;

  String lastName;

  int age;

  Person(this.firstName, this.lastName, this.age);

}

main() {

  Person p = new Person('mark', 'smith', 22);

  print('The persons name is ${p.firstName} ${p.lastName} and he is ${p.age}');

}

4- Raw Strings: in Dart, normally you can add escape characters to format your string. For example: ‘\n’ means ‘new line’. However, you can prefix the string with an ‘r’ to indicate to tell Dart to treat the string differently, to ignore escape characters.

main(){

print('this\nstring\nhas\nescape\ncharacters');

print('');

print(r'this\nstring\nhas\nescape\ncharacters');

}

5- ‘Dollar Sign’

void main() {

  double price = 100.75;

  print('Price is: \$${price}');

}

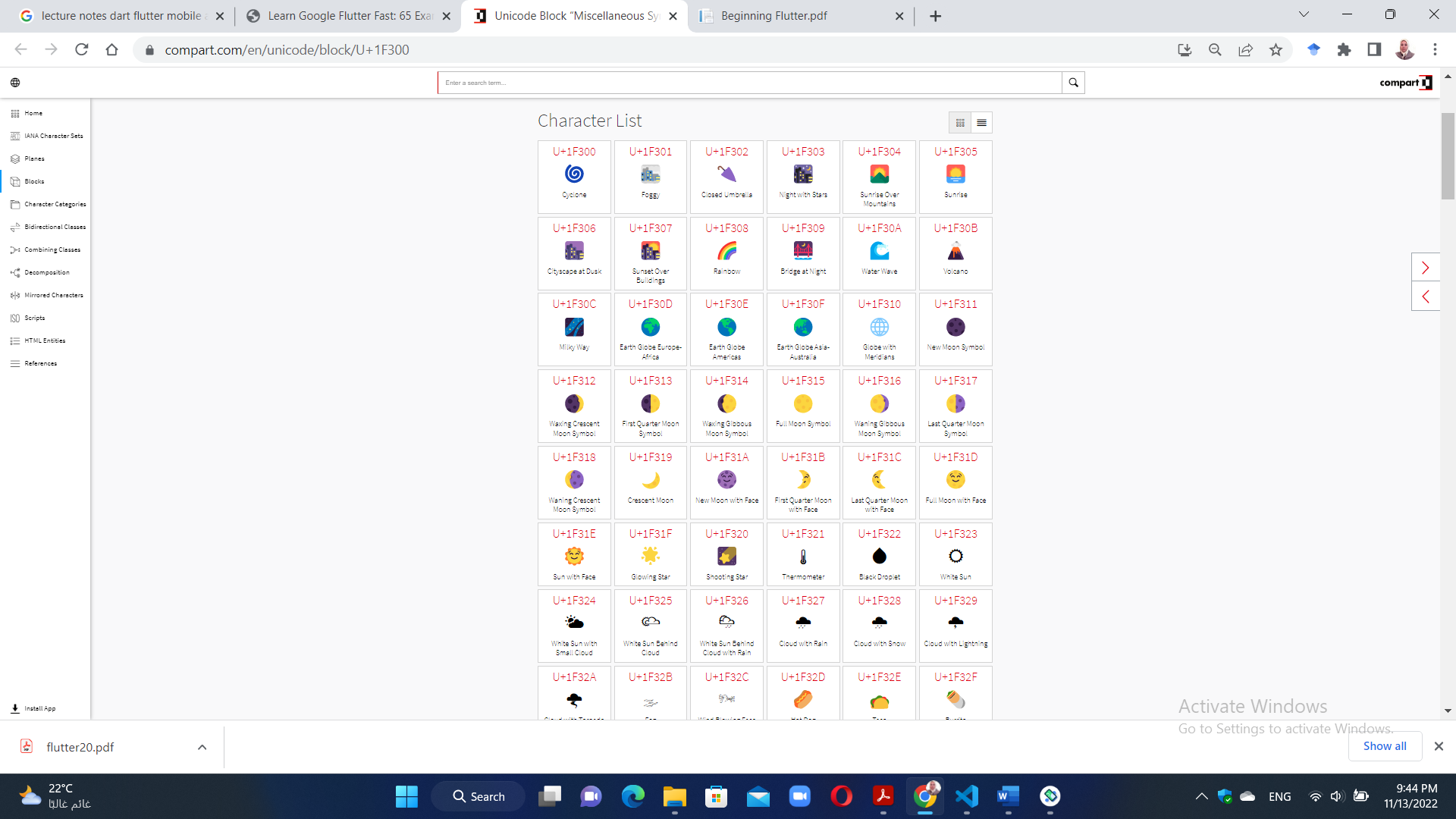
6- Runes are also special characters encoded into a string. Here is a link with a lot of the run codes: <https://www.compart.com/en/unicode/block/U+1F300>

main() {

  var clapping = '\u{1f44f}';

  print(clapping);

}



7- Constructors:

- Default Constructor: If you do not specify a constructor, a default constructor will be created for you without arguments. If you do specify a constructor, the default constructor won’t be created for you. -Constructor Syntax Shortcut: If you want to set the value of an instance variable in a constructor, you can use the ‘this.[instance variable name]’ to set it in the constructor signature.

class Name {

  String firstName;

  String lastName;

  Name(this.firstName, this.lastName);

}

main() {

  Name name = new Name('mark', 'smith');

  print(name.firstName);

  print(name.lastName);

}

8- Dart doesn’t need you to use the ‘new’ keyword when invoking constructors. However, you can keep it if you want.

void main() {

  Car car = Car("BMW", "M3");

  print(car.getBadge());

  Car car2 = new Car("BMW", "M3");

  print(car2.getBadge());

}

class Car {

  String \_make;

  String \_model;

  Car(this.\_make, this.\_model) {}

  String getBadge() {

    return \_make + " - " + \_model;

  }

}

9- Dart allows named constructors and I have found them very useful indeed if you want to instantiate the same class in different ways.

class ProcessingResult {

  bool \_error=true;

  String \_errorMessage='';

  ProcessingResult.success() {

    \_error = false;

    \_errorMessage = '';

  }

  ProcessingResult.failure(this.\_errorMessage) {

    //shortcut

    this.\_error = true;

  }

  String toString() {

    return 'Error: ' + \_error.toString() + ' Message: ' + \_errorMessage;

  }

}

void main() {

  print(ProcessingResult.success().toString());

  print(ProcessingResult.failure('it broke').toString());

}

10- s can help with the brevity of your code.

class Logger {

  void log(dynamic v) {

    print(DateTime.now().toString() + ' ' + v);

  }

}

main() {

// Without method cascades

  new Logger().log('program started');

  new Logger().log('doing something');

  new Logger().log('program finished');

// With method cascades

  new Logger()

    ..log('program started')

    ..log('going something')

    ..log('program finished');

}

11- This dart code creates a list then sorts it:

class Person {

  String \_firstName;

  String \_lastName;

  String \_phone;

  Person(this.\_firstName, this.\_lastName, this.\_phone);

  toString() {

    return "${\_firstName} ${\_lastName} ${\_phone}";

  }

}

void main() {

  List<Person> list = [

    Person("Mark", "Clow", "4043124462"),

    Person("Brant", "Sandermine", "4243124462"),

    Person("Phillip", "Perry", "4243124444")

  ];

  print("Not sorted: ${list}");

  list.sort((a, b) => a.\_firstName.compareTo(b.\_firstName));

  print("Sorted by first name: ${list}");

  list.sort((a, b) => a.\_firstName.compareTo(b.\_lastName));

  print("Sorted by last name: ${list}");

}

11- if statement

import 'dart:io';

void main() {

  var x = 10;

  if (x > 100)

    print("large value");

  else

    print("not large");

}

12- switch

import 'dart:io';

void main() {

  var x = 10;

  switch (x) {

    case 10:

      print("bad");

      break;

    case 20:

      print("small");

      break;

    default:

      print("good");

  }

}

13- for loop

import 'dart:io';

void main() {

  var x = 0, y = 0;

  for (int i = 20; i < 51; i++) {

    if (i % 2 == 0) x = x + i;

    if (x > 100) continue;

    else

      y = y + i;

    if (y > 100) break;

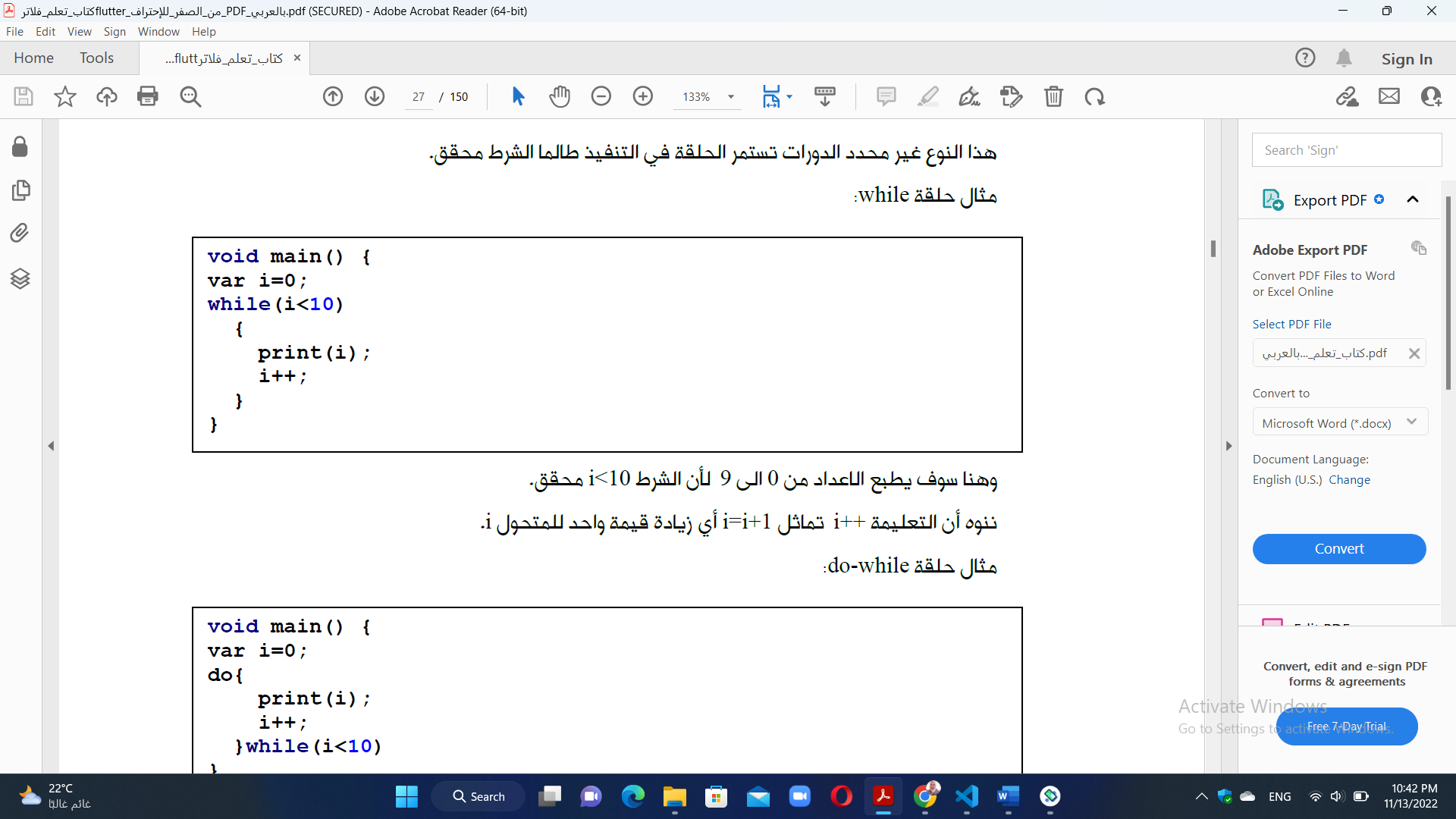
  }

  print(x);

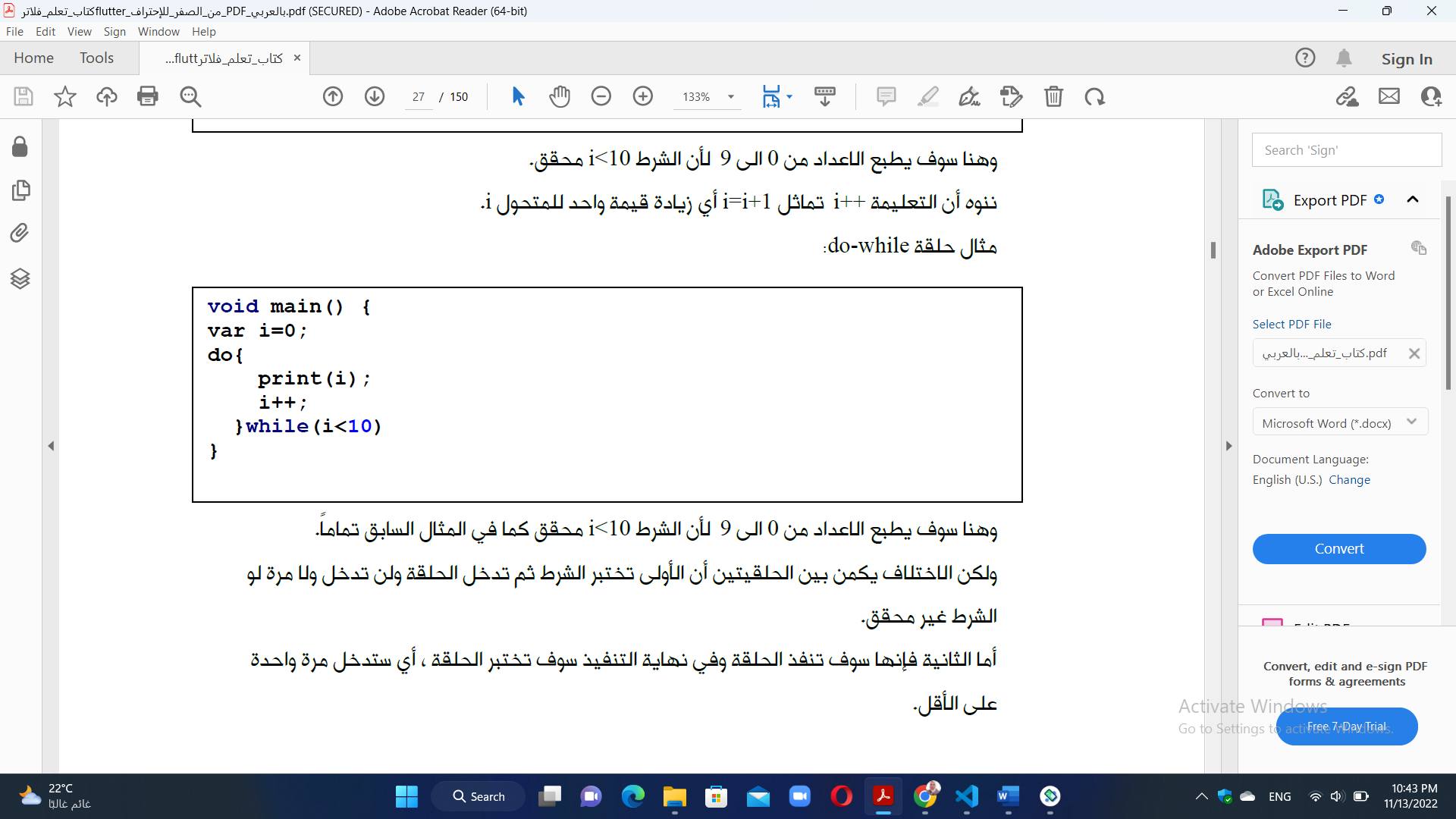
  print(y);

}

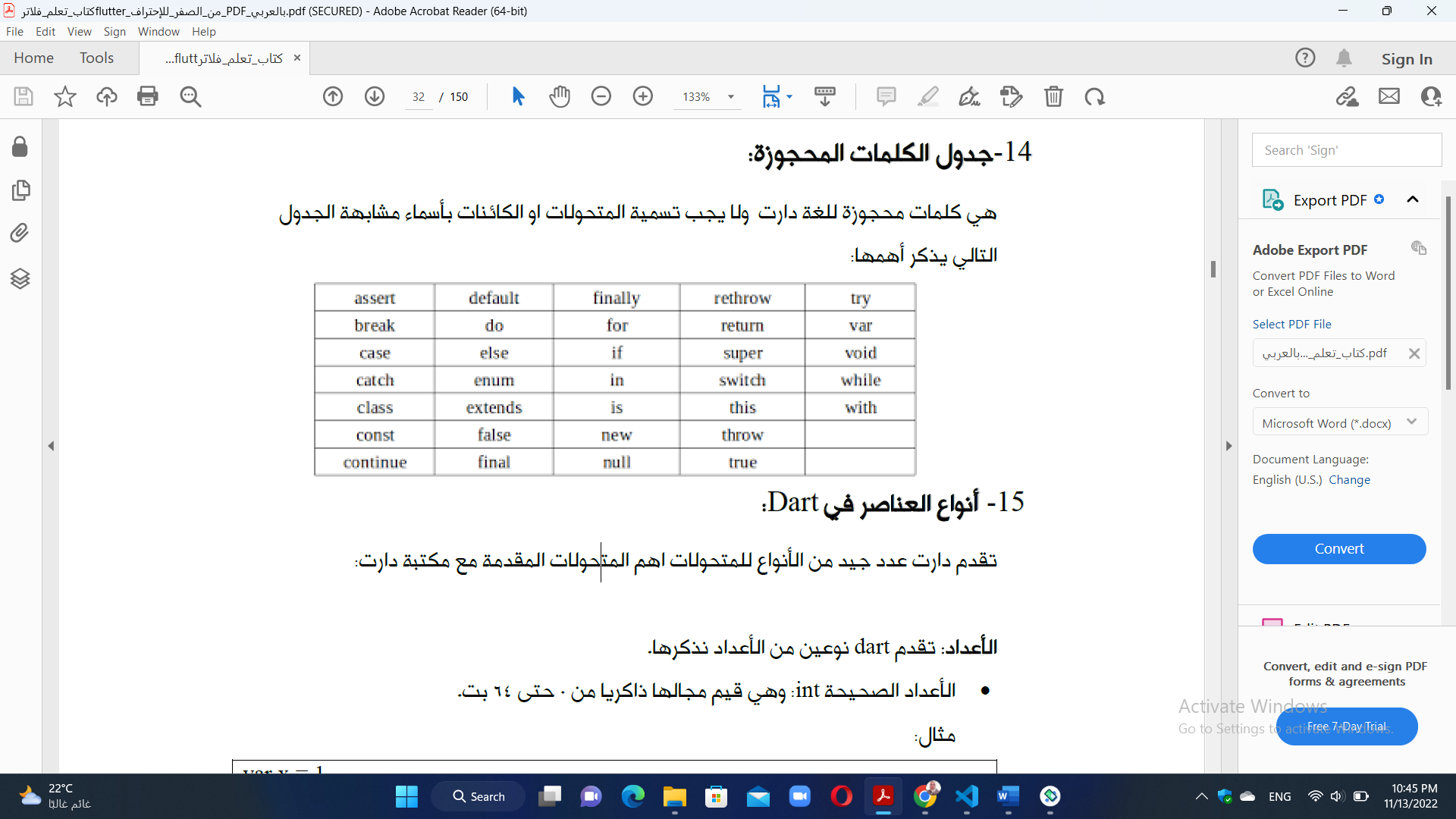
14-while



1. Do -while



1. Keywords



1. Try

In the following example, we have used the same code as above. The only difference is that the **catch block** (instead of the ON block) here contains the code to handle the exception. The parameter of **catch** contains the exception object thrown at runtime.

1. main() {
2. int x = 12;
3. int y = 0;
4. int res;
6. try {
7. res = x ~/ y;
8. }
9. catch(e) {
10. print(e);
11. }
12. }

Using On

main() {

   int x = 12;

   int y = 0;

   int res;

   try {

      res = x ~/ y;

   }

   on IntegerDivisionByZeroException catch(e) {

      print(e);

   }

}

Using finally

main() {

  int x = 12;

  int y = 0;

  int res = 0;

  try {

    res = x ~/ y;

  } on IntegerDivisionByZeroException {

    print('Cannot divide by zero');

  } finally {

    print('Finally block executed');

    print(res);

  }

}

The **throw** keyword is used to explicitly raise an exception. A raised exception should be handled to prevent the program from exiting abruptly.

The **syntax** for raising an exception explicitly is −

throw new Exception\_name()

example

main() {

  try {

    test\_age(-2);

  } catch (e) {

    print('Age cannot be negative');

  }

}

void test\_age(int age) {

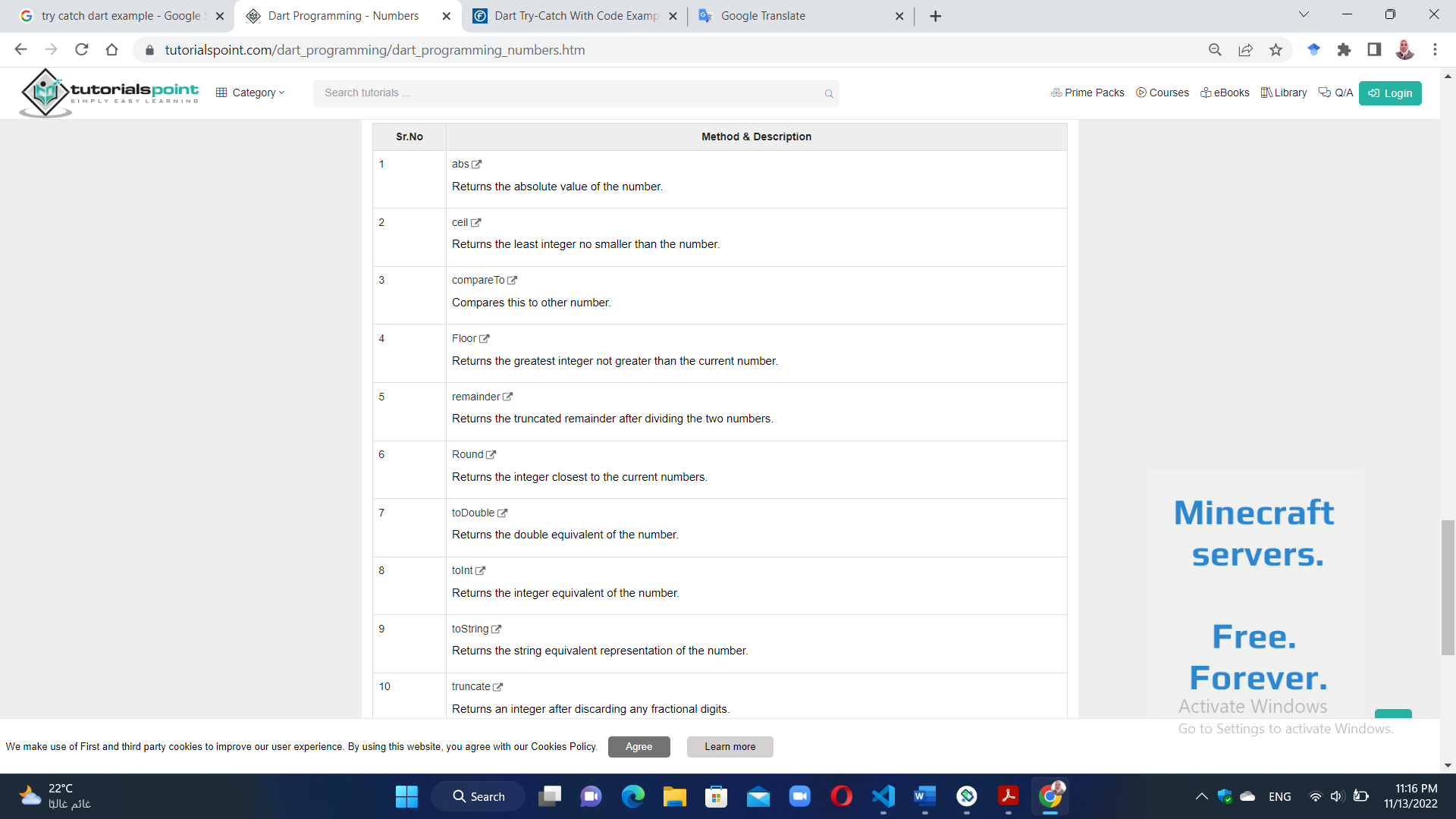
  if (age < 0) {

    throw new FormatException();

  }

}

-number method



mporting a library

Importing makes the components in a library available to the caller code. The import keyword is used to achieve the same. A dart file can have multiple import statements.

Built in Dart library URIs use the dart: scheme to refer to a library. Other libraries can use a file system path or the package: scheme to specify its URI. Libraries provided by a package manager such as the pub tool uses the *package: scheme*.

The syntax for importing a library in Dart is given below −

import 'URI'

Consider the following code snippet −

import 'dart:io'

import 'package:lib1/libfile.dart'

If you want to use only part of a library, you can selectively import the library. The syntax for the same is given below −

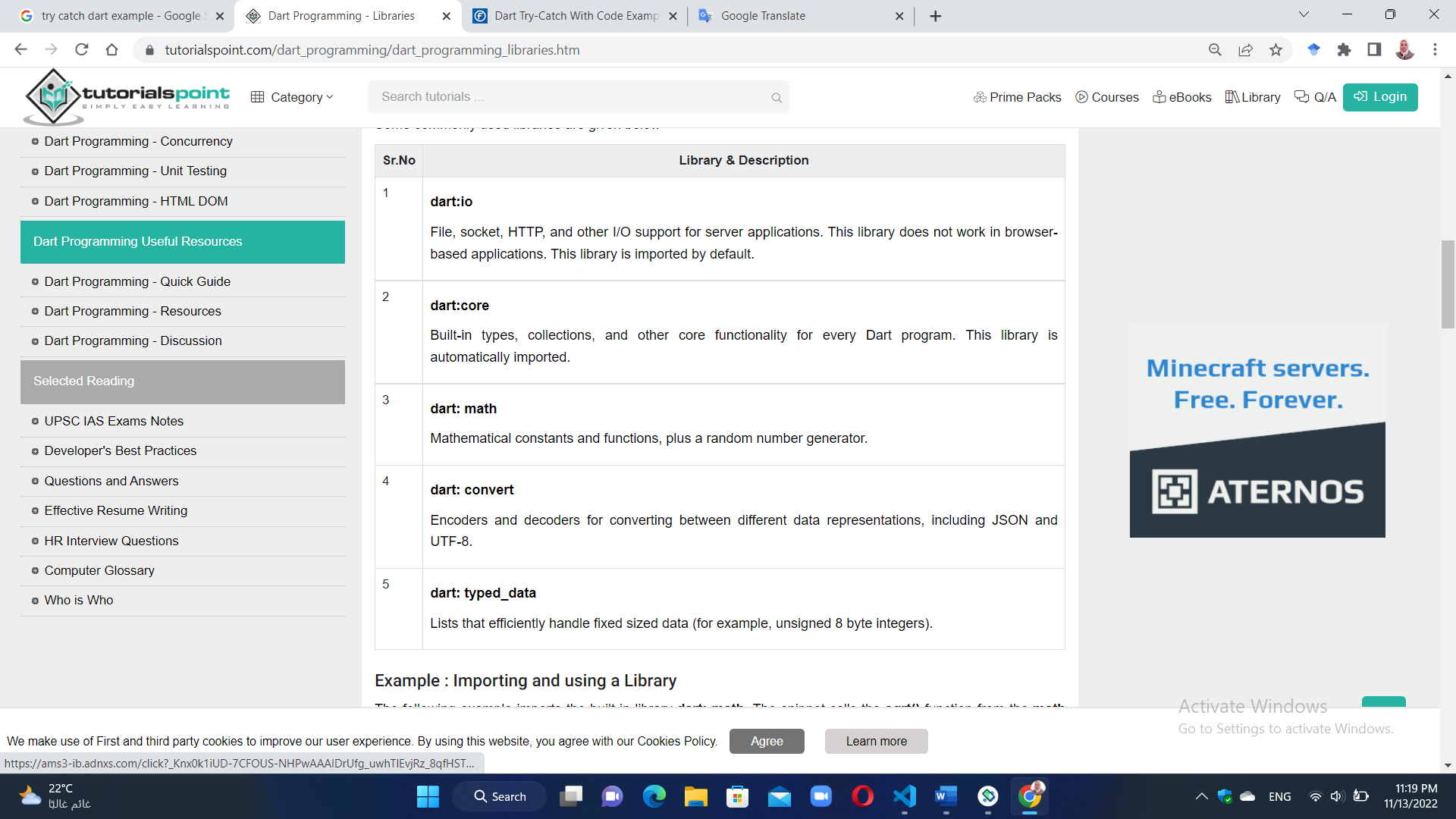
import 'package: lib1/lib1.dart' show foo, bar;

// Import only foo and bar.

import 'package: mylib/mylib.dart' hide foo;

// Import all names except foo

Some commonly used libraries are given below −



Example

import 'dart:math';

void main() {

print("Square root of 36 is: ${sqrt(36)}");

}