Recap – Functions and Pattern Matching

Partial Matches

- We have seen that a pattern matching block like ***{ case "ping" => "pong" }*** can be given type ***String => String***

```
val f: String => String = { case "ping" => "pong" }
```

- But the function is not defined on all its domain: ***f("pong")*** gives a ***MatchError***; that indicates that the pattern match lacks the definition for the case "pong", which throws an Exception

Partial Functions

- It would be nice to find beforehand given the function f, whether the function is applicable to a certain argument and unfortunately, with the function type itself, we can't do that
- We can give the expression "ping" to "pong" the type PartialFunction
- PartialFunction is a subtype of Function that besides applying it to an argument, it also allows you to query whether the function is defined for a given argument
- There is a way to find out whether the function can be applied to a given argument before running it:

```
val f: String => String = { case "ping" => "pong" }
f.isDefinedAt("ping") // true
f.isDefinedAt("pong") // false
```

- The partial function trait is defined as follows:

```
trait PartialFunction[-A, +R] extends Function1[-A, +R] {
   def apply(x: A): R
   def isDefinedAt(x: A): Boolean
}
```

- The apply method is inherited from the Function1 trait

Partial Function Objects

- If the expected type is a PartialFunction, the Scala compiler will expand { case "ping" => "pong" } as follows:

```
new PartialFunction[String, String] {
   def apply(x: String) = x match {
         case "ping" => "pong"
   }
   def isDefinedAt(x: String) = x match {
         case "ping" => true
         case _ => false
   }
}
```

Exercises

- Given the function

  ```
  val f: PartialFunction[List[Int], String] = {
      case Nil => "one"
      case x :: y :: rest => "two"
  }
  ```

  What do you expect is the result of ***f.isDefinedAt(List(1, 2, 3))*** ? – ***true****, the second pattern definitely does match a list of three elements. The first one would be x, the second y and the rest of the elements would be captured in the variable rest.*

- How about the following variation:

  ```
  val g: PartialFunction[List[Int], String] = {
      case Nil => "one"
      case x :: rest =>
          rest match {
          case Nil => "two"
      }
  }
  ```

  g.isDefinedAt(List(1, 2, 3)) gives: ***true****, the second case of g matches a list of three elements, though when you run this function, you would get a MatchError, because in the nested pattern match, the case of this list is not defined.*
  So what you see in this example is that ***the isDefinedAt method checks only the outermost matching block****. It is not a guarantee that if a function is defined at an argument, this function definitely will not throw a match error when it is run.*