Chapter 2

Control Structures and Functions

Printing

```
print("Answer:" + 42)
printf("Hello, %s! You are %d years old.%n", "Ia", 21)
print(f"Hello, $name! In six months, you'll be ${age + 0.5}%7.2f years old")
```

formatting

- %7.2f - formatted as a floating-point number of width 7 and precision 2
- With a prefix of s , strings can contain expressions but not format directives
- With a prefix of raw , escape sequences in a string are not evaluated. For example, raw"\n is a newline" starts with a backslash and the letter n, not a newline character
- Readline method takes prompt string

For Loops

```
for (c <- "Hello"; i <- 0 to 1) yield (c + i).toChar
 // Yields "HIeflmlmop"
for (i <- 0 to 1; c <- "Hello") yield (c + i).toChar
 // Yields Vector('H', 'e', 'l', 'l', 'o', 'l', 'f', 'm', 'm', 'p')
```

Default and Named Parameters

```
def decorate(str: String, left: String = "[", right: String = "]") =
 left + str + right
```

If you supply fewer arguments than there are parameters, the defaults are applied form the end

- named arguments need not be in the same order as the parameters

Variable arguments

```
def sum(args: Int*) = {
  var result = 0
  for (arg <- args) result += arg
  result
}
val s = sum(1 to 5) // Erro
val s = sum(1 to 5: _*) // Consider 1 to 5 as an argument sequence
```

tell the compiler that you want the parameter to be considered an argument sequence
This is needed in a recursive definition :

```
def recursiveSum(args: Int*) : Int = {
  if (args.length == 0) 0
  else args.head + recursiveSum(args.tail : _*)
}
```

Procedures

A procedure returns no value, and you only call it for its side effect. For example, the following procedure prints a string inside a box

```
def box(s : String) { // Look carefully: no =
  val border = "-" * (s.length + 2)
   print(f"$border%n|$s|%n$border%n")
}
```

Lazy Values

When a val is declared as lazy , its intiialization is deferred until it is accessed for the first time

- used to delay costly initialization statements
- deal with other initialization issues, such as circular dependencies
- it's not cost-free . Every time a lazy value is accessed, a method is called that checks whether a value has been initialized

Exceptions

- Scala has no checked exceptions - you have to excplicitly declare it for a function or method
- A throw expression has the special type Nothing
  Useful in if/else expressions. Type of the expression will be type of the other brach

```
val url = new URL("http://horstmann.com/fred-tiny.gif")
try {
  process(url)
} catch {
   case _: MalformedURLException => println(s"Bad URL: $url")
   case ex: IOException => ex.printStackTrace()
}
```