Functions and program organization

When a group of instructions need to be used many times in programs, *functions* save time in programming. Suppose you have a program that has these statements.

In [ ]:
```
println("If you're happy and you know it clap your hands.")
println("If you're happy and you know it clap your hands.")
println("If you're happy and you know it, and you really want to show it,")
println("If you're happy and you know it clap your hands.")
```

There is repetition, not too much. A function can abstract the common elements and state them in one place. Here is the definition of a function `happy`, starting with `fun`, that helps print the repetitive verses.

In [3]:
```
fun happy() = println("If you're happy and you know it clap your hands.")
happy()
happy()
println("If you're happy and you know it, and you really want to show it,")
happy()
```

...

This has not saved any code, but if it has centralized some of the work. We can look at the pattern in another way, and see it as a repetition of "If you're happy and you know it" with some suffix. The function could have a *parameter* that is the variable part of the message, that otherwise is the same. Here is the definition of the function `happy` with a parameter `suffix` that is printed as part of the message.

In [6]:
```
fun happy(suffix : String) = println("If you're happy and you know it$suffix")
happy(" clap your hands.")
happy(" clap your hands.")
happy(", and you really want to show it,")
happy(" clap your hands.")
```

...

We can introduce a second function to reduce some more repetition.

In [ ]:
```
fun happy(suffix : String) = println("If you're happy and you know it$suffix")
fun clap() = happy(" clap your hands.")
clap()
clap()
happy(", and you really want to show it,")
clap()
```

Again, not a lot of economy, but perhaps a desirable consistency.

We can see the same kind of repetition in expressions.

In [7]:
```
println("the square of 2 is ${2*2}")
println("the square of 3 is ${3*3}")
println("the square of 4 is ${4*4}")
```

...

Some tidiness can be achieved with a function that can return a value.

In [8]:
```
fun square(n : Int) = n * n
println("the square of 2 is ${square(2)}")
println("the square of 3 is ${square(3)}")
println("the square of 4 is ${square(4)}")
```

...

And regularizing some more.

In [9]:

```
fun square(n : Int) = n * n
fun show_square(n : Int) {
    println("the square of $n is ${square(n)}")
}
show_square(2)
show_square(3)
show_square(4)
```

...

The goal is not just shortening the program, but making it more *maintainable*. As programs evolve, changes that affect repetitive lines must be made in many places and are more prone to errors. The last examples above have abstracted the common elements into one place. There is a phrase in programming, *Don't repeat yourself*, that embodies this in the acronym *DRY*.