Flow Control

In [3]:
```
val age = 10

val y = if(age < 65) {
    "Not a senior"
} else {
    "Senior"
}

println(y)
```

```
Not a senior
```

When: A better if-else

In [9]:
```
val z = when(y) {
    "Senior" -> "Must be over 65"
    "Not a senior" -> "Must be under 65"
    else -> "Don't know..."
}
```

In [10]:
```
z
```

Out[10]:
```
Must be under 65
```

In [14]:
```
// Pattern matching for ranges

val age = 4
when(age) {
    in 0 .. 10 -> "Child"
    in 10 .. 20 -> "Teen"
    in 20 .. 65 -> "Adult"
    else -> "Senior"
}
```

Out[14]:
```
Child
```

In [19]:
```
//        Number
//       /  |  \
//   Int, Float, Double

val num: Number = 3

when(num) {
    is Int -> "$num: Int"
    is Float -> "$num: Float"
    is Double -> "$num: Double"
    else -> "$num is unknown type"
}
```

Out[19]:
```
3: Int
```

Integer ranges

In [20]:
```
1..10
```

Out[20]:
```
1..10
```

In [21]:
```
(1..10).toList()
```

Out[21]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [22]:
```
1 until 10
```

Out[22]:
```
1..9
```

In [23]:
```
(1 until 10).toList()
```

Out[23]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [25]:
```
(10 downTo 1).toList()
```

Out[25]:
```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

For-loop over ranges

In [26]:
```
for(i in 1..10) {
    println("i = $i")
}
```

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

Top level function declaration

In [27]:
```
// function declaration with explicit body block

fun factorial(n: Int): Int {
    if(n == 0) {
        return 1
    } else {
```

```
        return n * factorial(n-1)
    }
}
```

In [28]:
```
factorial(10)
```

Out[28]:
```
3628800
```

In [29]:
```
// function declaration with = return expression

fun factorial(n: Int): Int = if(n == 0) 1 else { n * factorial (n-1) }
```

In [30]:
```
factorial(10)
```

Out[30]:
```
3628800
```

Extension functions

In [31]:
```
fun String.reverse(): String {
    var result = ""
    for(i in (this.length-1 downTo 0)) {
        result += this.get(i)
    }
    return result
}
```

In [35]:
```
"1234".reverse()
```

Out[35]:
```
4321
```

Infix operators

In [40]:
```
infix fun String.repeat(n: Int): String {
    var result = ""
    for(i in 1..n) {
        result += " " + this
    }
    return result
}
```

In [45]:
```
"Hello" repeat 3 repeat 3
```

Out[45]:
```
 Hello Hello Hello  Hello Hello Hello  Hello Hello Hello
```

Functions as values using anonymous functions

In [46]:
```
val f = fun(x:String, repeats: Int): String = x repeat repeats
```

In [47]:
```
f("Hello", 2)
```

Out[47]:
```
 Hello Hello
```

Anonymous function as block containing parameters

In [48]:
```
val f = {
    x:String, n: Int -> x repeat n
}
```

In [49]:
```
f("Hello", 3)
```

Out[49]:
```
 Hello Hello Hello
```

Anonymous function as block containing `it`

In [50]:
```
val repeat3Times: (String) -> String = { it repeat 3 }
```

In [51]:
```
repeat3Times("Hello")
```

Out[51]:
```
 Hello Hello Hello
```

Lists (immutable)

In [54]:
```
val xs: List<String> = listOf<String>("a", "b", "c")
xs
```

Out[54]:
```
[a, b, c]
```

In [56]:
```
val xs = listOf<String>("x", "y", "z")
xs
```

Out[56]:
```
[x, y, z]
```

In [58]:
```
val xs = listOf("hello", "world", "again")
xs
```

Out[58]:
```
[hello, world, again]
```

In [61]:
```
listOf("hello", 1, 2, 3, 3.1415f)
```

Out[61]:
```
[hello, 1, 2, 3, 3.1415]
```

In [63]:
```
val xs = listOf("a", "b", "c")
xs
```

Out[63]: [a, b, c]

In [64]:
```
xs + "x"
```

Out[64]: [a, b, c, x]

In [65]:
```
xs
```

Out[65]: [a, b, c]

In [66]:
```
xs - "b"
```

Out[66]: [a, c]

In [67]:
```
xs
```

Out[67]: [a, b, c]

In [68]:
```
xs.drop(1)
```

Out[68]: [b, c]

In [69]:
```
xs
```

Out[69]: [a, b, c]

Mutable Lists

In [70]:
```
val xs = mutableListOf("a", "b", "c")
xs
```

Out[70]: [a, b, c]

In [71]:
```
xs += "x"
xs
```

Out[71]: [a, b, c, x]

In [73]:
```
xs += listOf("1", "2", "3")
```

In [74]:
```
xs
```

Out[74]: [a, b, c, x, 1, 2, 3]

In [ ]: