스칼라 프로그래밍 언어 기초

순천향대학교 컴퓨터공학과

이 상 정

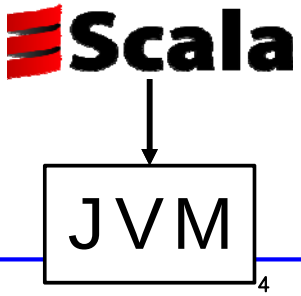
학습 내용

1. 스칼라 소개 및 설치
2. 스칼라 기초
3. 함수
4. 클래스
5. 객체, 케이스 클래스, 트레이트
6. 컬렉션

1. 스칼라 소개 및 설치

스칼라 (Scala) 언어

- Scala = scalable(확장 가능한) 언어라는 뜻을 가짐
  - 객체지향 + 함수형 프로그래밍 언어 (functional programming language)
    - 함수형 프로그래밍 언어는 순수 함수(pure function) 작성 가능
      - 같은 입력이 주어지면, 항상 같은 출력을 반환
      - 부작용 없음 (side effect)
  - 기존 자바(Java)의 복잡함을 극복하기 위해 2004년 M. Odersky가 개발
  - 스칼라 언어 컴파일하면 자바 바이트코드 생성
    - JVM 상에서 동작
  - 자바의 문법과 라이브러리 그대로 사용 가능
    - Java → Scala 변환 가능 (대부분)
    - Java ← Scala 변환 가능 (제한적)



Scala 설치 – 리눅스 (1)

- 우분투 18.04에 루트 권한으로 스칼라 설치
  - https://www.scala-lang.org/
  - Scala 2.12.11 버전 설치
    - 마스터 서버에 설치
    - 최근 버전은 2.13.1(2020년 3월 기준)이나 스파크의 스칼라 버전과 같은 버전 설치

```
$ cd /usr/local/src
$ sudo wget https://downloads.lightbend.com/scala/2.12.11/scala-2.12.11.tgz
$ sudo mkdir scala
$ sudo tar -xvf scala-2.12.11.tgz -C scala
```

  - ~/.bashrc 파일의 PATH 환경변수에 스칼라 경로 추가

```
.......
export SCALA_HOME=/usr/local/src/scala/scala-2.12.11
export PATH=$SCALA_HOME/bin:$PATH
```

```
$ source ~/.bashrc
```


Scala 실행 – 리눅스

```
bigdata@master:~$ scala
Welcome to Scala 2.12.11 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_201).
Type in expressions for evaluation. Or try :help.

scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:completions <string>    output completions for the given string
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicits [-v]          show the implicits in scope
:javap <path|class>      disassemble a file or class name
:line <id>|<line>        place line(s) at the end of history
:load <path>             interpret lines in a file
:paste [-raw] [path]     enter paste mode or paste a file
:power                   enable power user mode
:quit                    exit the interpreter
:replay [options]        reset the repl and replay all previous commands
:require <path>          add a jar to the classpath
:reset [options]         reset the repl to its initial state, forgetting all session entries
:save <path>             save replayable session to a file
:sh <command line>       run a shell command (result is implicitly => List[String])
:settings <options>      update compiler options, if possible; see reset
:silent                  disable/enable automatic printing of results
:type [-v] <expr>        display the type of an expression without evaluating it
:kind [-v] <type>        display the kind of a type. see also :help kind
:warnings                show the suppressed warnings from the most recent line which had any

scala> :quit
bigdata@master:~$
```

SBT 설치

SBT (Simple Build Tool)

- Scala 기반 빌드 툴로 Java, Scala 소스 코드 빌드
  - 아파치 Ivy 사용하여 의존성(dependency) 관리
  - Apache Maven 보다는 단순하고 쉽게 사용
  - 로컬 저장 소 디렉토리: ~/ivy2/cache
  - http://www.scala-sbt.org/
- 마스터에 설치
  - https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html 참조
  - org.scala-sbt sbt 1.3.8 (2020년 3월 기준)

```
$ echo "deb https://dl.bintray.com/sbt/debian /" |
  sudo tee -a /etc/apt/sources.list.d/sbt.list
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
  2EE0EA64E40A89B84B2DF73499E82A75642AC823
$ sudo apt-get update
$ sudo apt-get install sbt
```



```
bigdata@ ~:~$ echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
deb https://dl.bintray.com/sbt/debian /
bigdata@slave1:~$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF73499E82A75642AC823
Executing: /tmp/apt-key-gpghome.LZmTVqHspb/gpg.1.sh --keyserver hkp://keyserver.ubuntu.com:80 --recv 2EE0EA64E40A89B84B2DF7349
642AC823
gpg: key 99E82A75642AC823: public key "sbt build tool <scalasbt@gmail.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
bigdata@slave1:~$ sudo apt-get update
Ign:1 https://dl.bintray.com/sbt/debian  InRelease
Get:2 https://dl.bintray.com/sbt/debian  Release [815 B]
Get:3 https://dl.bintray.com/sbt/debian  Release.gpg [821 B]
Get:4 https://dl.bintray.com/sbt/debian  Packages [3,390 B]
Hit:5 http://archive.ubuntu.com/ubuntu bionic InRelease
Hit:6 http://archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:7 http://archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:8 http://archive.ubuntu.com/ubuntu bionic-security InRelease
Fetched 5,026 B in 2s (2,136 B/s)
Reading package lists... Done
bigdata@slave1:~$ sudo apt-get install sbt
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  ca-certificates-java default-jre-headless fontconfig-config fonts-dejavu-core java-common libasound2 libasound2-data
  libavahi-client3 libavahi-common-data libavahi-common3 libcups2 libfontconfig1 libhawtjni-runtime-java libjansi-java

Need to get 1,126 kB of archives.
After this operation, 1,303 kB of additional disk space will be used.
Get:1 https://dl.bintray.com/sbt/debian  sbt 1.2.8 [1,126 kB]
Fetched 1,126 kB in 3s (428 kB/s)
Selecting previously unselected package sbt.
(Reading database ... 67904 files and directories currently installed.)
Preparing to unpack .../apt/archives/sbt_1.2.8_all.deb ...
Unpacking sbt (1.2.8) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up sbt (1.2.8) ...
Creating system group: sbt
Creating system user: sbt in sbt with sbt daemon-user and shell /bin/false
bigdata@s~:~$
```

SBT 실행

- 처음 실행 시 저장소 라이브러리 적재

  $ sbt

```
bigdata@slave1:~$ sbt
Getting org.scala-sbt sbt 1.2.8  (this may take some time)...
downloading https://repo1.maven.org/maven2/org/scala-sbt/sbt/1.2.8/sbt-1.2.8.jar ...
        [SUCCESSFUL ] org.scala-sbt#sbt;1.2.8!sbt.jar (698ms)
downloading https://repo1.maven.org/maven2/org/scala-lang/scala-library/2.12.7/scala-library-2.12.7
        [SUCCESSFUL ] org.scala-lang#scala-library;2.12.7!scala-library.jar (3887ms)
downloading https://repo1.maven.org/maven2/org/scala-sbt/main_2.12/1.2.8/main_2.12-1.2.8.jar ...
        [SUCCESSFUL ] org.scala-sbt#main_2.12;1.2.8!main_2.12.jar (2429ms)
```

```
[info] downloading https://repo1.maven.org/maven2/org/apache/logging/log4j/log4j-core/2.11.1/log4j
[info]  [SUCCESSFUL ] org.apache.logging.log4j#log4j-core;2.11.1!log4j-core.jar(test-jar) (3333ms)
[info] Done updating.
[info] Set current project to bigdata (in build file:/home/bigdata/)
[info] sbt server started at local:///home/bigdata/.sbt/1.0/server/e1a29e31540551dd0cac/sock
sbt:bigdata>
```


윈도우 스칼라 설치 (1)

- 윈도우용 SBT 1.2.8 설치
  - http://www.scala-lang.org/download/ 에서 다운로드 설치
    - https://piccolo.link/sbt-1.2.8.msi
  - 설치 후, 윈도우 명령 프롬프트(cmd)에서 REPL (Read-Evaluate-Print Loop)
    - 인터프리터
  - sbt console 명령으로 실행

윈도우 스칼라 설치 (2)

```
명령 프롬프트 - sbt console - sbt console - sbt console

C:\Users\Yun>sbt console
Java HotSpot(TM) 64-Bit Server VM warning: Ignoring option MaxPermSize; support was removed in 8.0

Java HotSpot(TM) 64-Bit Server VM warning: Ignoring option MaxPermSize; support was removed in 8.0

[info] Loading settings from idea.sbt ...
[info] Loading global plugins from C:\Users\Yun\.sbt\1.0\plugins
[info] Loading project definition from C:\Users\Yun\project
[info] Set current project to yun (in build file:/C:/Users/Yun/)
[info] Starting scala interpreter...
Welcome to Scala 2.12.4 (Java HotSpot(TM) 64-Bit Server VM, Java 9.0.1).
Type in expressions for evaluation. Or try :help.

scala> print("hello World!")
hello World!
scala>
```


윈도우 스칼라 설치 (3)

```
명령 프롬프트

scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
:imports [name name ...] show import history, identifying sources of names
:implicits [-v]          show the implicits in scope
:javap <path|class>      disassemble a file or class name
:line <id>|<line>        place line(s) at the end of history
:load <path>             interpret lines in a file
:paste [-raw] [path]     enter paste mode or paste a file
:power                   enable power user mode
:quit                    exit the interpreter
:replay [options]        reset the repl and replay all previous commands
:require <path>          add a jar to the classpath
:reset [options]         reset the repl to its initial state, forgetting all session entries
:save <path>             save replayable session to a file
:sh <command line>       run a shell command (result is implicitly => List[String])
:settings <options>      update compiler options, if possible; see reset
:silent                  disable/enable automatic printing of results
:type [-v] <expr>        display the type of an expression without evaluating it
:kind [-v] <type>        display the kind of a type. see also :help kind
:warnings                show the suppressed warnings from the most recent line which had any

scala> :quit

[success] Total time: 72 s, completed 2018. 3. 15 오전 11:05:03

C:₩Users₩sjlee>
```

2. 스칼라 기초

표현식 (Expression)

- 표현식 (expression)은 실행 후 값을 반환하는 단위

```
scala> 5 * 7
res0: Int = 35

scala>
```

```
scala> 2 * res0
res1: Int = 70

scala>
```

- 표현식을 입력하면 인터프리터가 결과를 출력
- res0는 결과 값에 인터프리터가 부여한 이름
  - 이 값의 타입은 Int이고, 값은 35라는 정수

```
scala> val amount = { val x = 5 * 20; x + 10 }
amount: Int = 110
```

```
scala> val amount = {
     |   val x = 5 * 20
     |   x + 10
     | }
amount: Int = 110
```

값 (Value)

- 값은 val 로 정의하며 변경 불가능

```
val <identifier>[: <type>] = <data>
```

```
scala> val x: Int = 20
x: Int = 20

scala> val greeting: String = "Hello, World"
greeting: String = Hello, World

scala> val atSymbol: Char = '@'
atSymbol: Char = @
```

```
scala> val x = 20
x: Int = 20

scala> val greeting = "Hello, World"
greeting: String = Hello, World

scala> val atSymbol = '@'
atSymbol: Char = @
```


변수 (Variable)

- 변수는 var 로 정의
  - 값이 변경되지 않는 경우 val을 우선적으로 사용 권장

```
var <identifier>[: <type>] = <data>
```

```
scala> var x = 5
x: Int = 5

scala> x = x * 4
x: Int = 20
```

```
scala> var y = 1.5
y: Double = 1.5

scala> y = 42
y: Double = 42.0
```

```
scala> var x = 5
x: Int = 5

scala> x = "what's up?"
<console>:8: error: type mismatch;
 found   : String("what\'s up?")
 required: Int
       x = "what's up?"
           ^
```

기본 타입 (Basic Type)

❑ 스칼라의 기본 타입

- to<Type> 메서드 사용하여 수동으로 형 변환

Value type	Range
Byte	8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive)
Short	16-bit signed two's complement integer (-2^{15} to $2^{15} - 1$, inclusive)
Int	32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive)
Long	64-bit signed two's complement integer (-2^{63} to $2^{63} - 1$, inclusive)
Char	16-bit unsigned Unicode character (0 to $2^{16} - 1$, inclusive)
String	a sequence of Chars
Float	32-bit IEEE 754 single-precision float
Double	64-bit IEEE 754 double-precision float
Boolean	true or false

기본 타입 예

```
scala> val b: Byte = 10
b: Byte = 10

scala> val s: Short = b
s: Short = 10

scala> val d: Double = s
d: Double = 10.0
```

```
scala> val l: Long = 20
l: Long = 20

scala> val i: Int = l
<console>:8: error: type mismatch;
 found   : Long
 required: Int
       val i: Int = l
```

```
scala> val l: Long = 20
l: Long = 20

scala> val i: Int = l.toInt
i: Int = 20
```

```
scala> val hello = "Hello There"
hello: String = Hello There

scala> val signature = "With Regards, \nYour friend"
signature: String =
With Regards,
Your friend
```

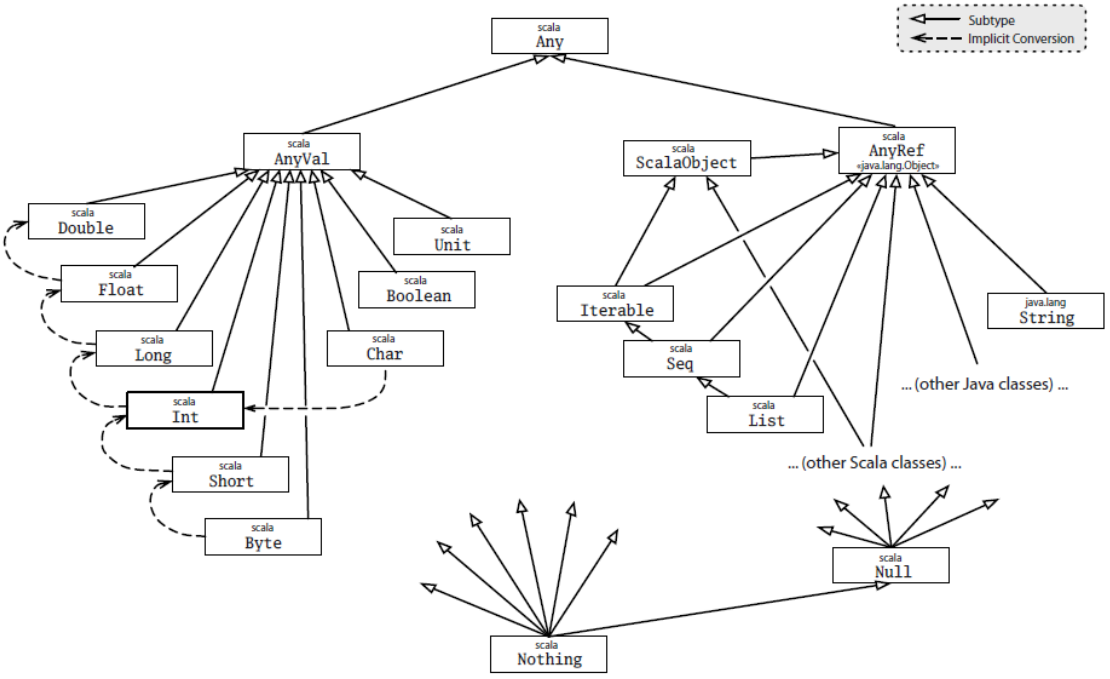
```
scala> val greeting = "Hello, " + "World"
greeting: String = Hello, World

scala> val matched = (greeting == "Hello, World")
matched: Boolean = true

scala> val theme = "Na " * 16 + "Batman!" // what do you expect this to print?
```

스칼라 타입 계층 구조

- 스칼라의 모든 자료의 타입은 계층 구조의 클래스로 정의



문자열 보간 (String Interpolation)

- 문자열 상수 안에서 변수의 값을 기술
  - 문자열 안의 $는 언어의 %s같은 기능으로 문자열 합성
  - { }로 감싸면, 식의 결과를 표시
  - 문자열 상수는 s로 시작

```
scala> val approx = 355/113f
approx: Float = 3.141593

scala> println("Pi, using 355/113, is about " + approx + "." )
Pi, using 355/113, is about 3.141593.
```

```
scala> println(s"Pi, using 355/113, is about $approx." )
Pi, using 355/113, is about 3.141593.
```

```
scala> val item = "apple"
item: String = apple

scala> s"How do you like them ${item}s?"
res0: String = How do you like them apples?

scala> s"Fish n chips n vinegar, ${"pepper "*3}salt"
res1: String = Fish n chips n vinegar, pepper pepper pepper salt
```

조건문 – If문

If Expressions

```
if (<Boolean expression>) <expression>
```

```
scala> if ( 47 % 3 > 0 ) println("Not a multiple of 3")
Not a multiple of 3
```

```
scala> val result = if ( false ) "what does this return?"
result: Any = ()
```

If-Else Expressions

```
if (<Boolean expression>) <expression>
else <expression>
```

```
scala> val x = 10; val y = 20
x: Int = 10
y: Int = 20

scala> val max = if (x > y) x else y
max: Int = 20
```


조건문 – 매치문 (Match)

- match 키워드는 C의 switch와 유사

```
<expression> match {
  case <pattern match> => <expression>
  [case...]
}
```

```
scala> val x = 10; val y = 20
x: Int = 10
y: Int = 20

scala> val max = x > y match {
     |   case true => x
     |   case false => y
     | }
max: Int = 20
```

```
scala> val status = 500
status: Int = 500

scala> val message = status match {
     |     case 200 =>
     |         "ok"
     |     case 400 => {
     |         println("ERROR - we called the service incorrectly")
     |         "error"
     |     }
     |     case 500 => {
     |         println("ERROR - the service encountered an error")
     |         "error"
     |     }
     | }
ERROR - the service encountered an error
message: String = error
```

```
scala> val message = "Ok"
message: String = Ok

scala> val status = message match {
     |   case "Ok" => 200
     |   case other => {
     |     println(s"Couldn't parse $other")
     |     -1
     |   }
     | }
status: Int = 200
```

루프

```
while (<Boolean expression>) statement
```

```
scala> var x = 10; while (x > 0) x -= 1
x: Int = 0
```

```
scala> val x = 0
x: Int = 0

scala> do println(s"Here I am, x = $x") while (x > 0)
Here I am, x = 0
```

```
for (<identifier> <- <iterator>) [yield] [<expression>]
```

- <iterator > = `<starting integer> [to|until] <ending integer> [by increment]`
- yield 는 컬렉션으로 반환

```
scala> for (x <- 1 to 7) { println(s"Day $x:") }
Day 1:
Day 2:
Day 3:
Day 4:
Day 5:
Day 6:
Day 7:
```

```
scala> for (x <- 1 to 7) yield { s"Day $x:" }
res10: scala.collection.immutable.IndexedSeq[String] = Vector(Day 1:,
Day 2:, Day 3:, Day 4:, Day 5:, Day 6:, Day 7:)
```


반복자 가드 (Iterator Guard)

- 반복자 가드 (Iterator Guard) 또는 필터 (filter)는 if 표현식이 참일 때만 반복을 수행

```
for (<identifier> <- <iterator> if <Boolean expression>) ...
```

```
scala> val threes = for (i <- 1 to 20 if i % 3 == 0) yield i
threes: scala.collection.immutable.IndexedSeq[Int] = Vector(3, 6, 9, 12, 15, 18)
```

```
scala> val quote = "Faith,Hope,,Charity"
quote: String = Faith,Hope,,Charity

scala> for {
     |   t <- quote.split(",")
     |   if t != null
     |   if t.size > 0
     | }
     | { println(t) }
Faith
Hope
Charity
```

3. 함수

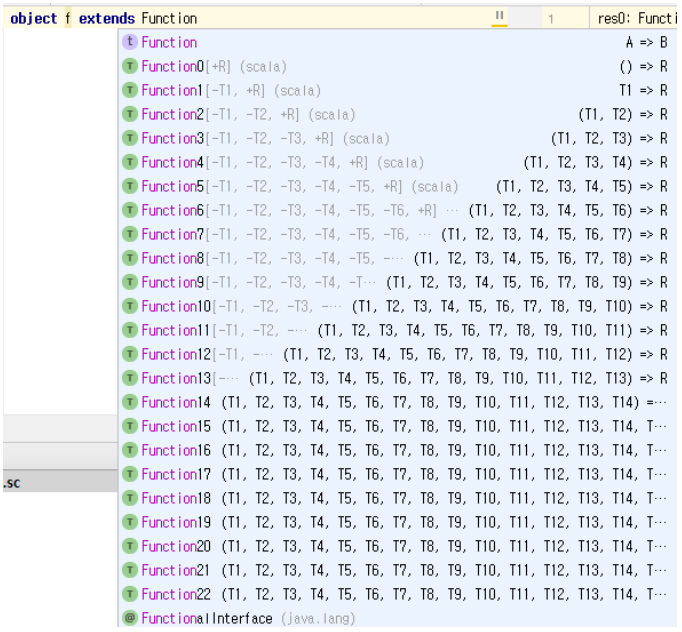
스칼라 함수 (1)

- 스칼라는 함수형 프로그래밍 언어로 구성력이 좋은 함수의 정의 지원
  - 표준 함수형 프로그래밍의 방법론(재사용성)을 따르고 가능하면 순수 함수(pure function)를 구성하면 함수형 언어의 이점을 활용할 수 있음

- 순수 함수 : 부작용(side-effect)이 없는 함수
  - 함수의 결과가 외부의 영향을 받지 않고, 오직 입력에 의해서 결정
    - 외부의 영향 : 입출력(IO), 전역 변수등
    - 동일 입력에 대해 항상 같은 값을 반환

스칼라 함수 (2)

- 스칼라에선 함수도 하나의 객체 (값) 로 구현
  - 0-22개의 매개 변수를 입력받는 객체로 구현



함수 정의 (1)

- def 로 함수 정의
  - return 키워드가 없음 → 마지막 문장의 값 반환
  - 반환 값이 없을 때 → Unit → return void

```
def <identifier> = <expression>
```

```
scala> def hi = "hi"
hi: String

scala> hi
res0: String = hi
```

```
def <identifier>: <type> = <expression>
```

```
scala> def hi: String = "hi"
hi: String
```

```
def <identifier>(<identifier>: <type>[, ... ]): <type> = <expression>
```

```
scala> def multiplier(x: Int, y: Int): Int = { x * y }
multiplier: (x: Int, y: Int)Int

scala> multiplier(6, 7)
res0: Int = 42
```

함수 정의 (2)

```
scala> def power(x: Int, n: Int): Long = {
     |   if (n >= 1) x * power(x, n-1)
     |   else 1
     | }
power: (x: Int, n: Int)Long

scala> power(2, 8)
res6: Long = 256

scala> power(2, 1)
res7: Long = 2
```

VarArg Parameters

```
scala> def sum(items: Int*): Int = {
     |   var total = 0
     |   for (i <- items) total += i
     |   total
     | }
sum: (items: Int*)Int

scala> sum(10, 20, 30)
res11: Int = 60

scala> sum()
res12: Int = 0
```

Calling Functions With Named Parameters

```
<function name>(<parameter> = <value>)
```

```
scala> def greet(prefix: String, name: String) = s"$prefix $name"
greet: (prefix: String, name: String)String

scala> val greeting1 = greet("Ms", "Brown")
greeting1: String = Ms Brown

scala> val greeting2 = greet(name = "Brown", prefix = "Mr")
greeting2: String = Mr Brown
```


타입 매개변수 (Type Parameter)

- 값 매개변수 또는 반환값에 사용될 타입을 지시하는 타입 매개변수(type parameter)를 전달

```
def <function-name>[type-name](parameter-name>: <type-name>): <type-name>...
```

```
def identity(s: String): String = s
```

```
def identity(i: Int): Int = i
```

```
scala> def identity(a: Any): Any = a
identity: (a: Any)Any

scala> val s: String = identity("Hello")
<console>:8: error: type mismatch;
 found   : Any
 required: String
       val s: String = identity("Hello")
                                ^
```

```
scala> def identity[A](a: A): A = a
identity: [A](a: A)A

scala> val s: String = identity[String]("Hello")
s: String = Hello

scala> val d: Double = identity[Double](2.717)
d: Double = 2.717
```

일급 함수

- 함수형 프로그래밍 언어에서는 함수는 일급 함수(first-class function)이어야 함
  - 함수는 다른 데이터 타입과 마찬가지로 값, 변수 등에 저장
  - 다른 함수의 매개 변수로 사용되거나 반환값으로 사용
    => 고차 함수

- 다른 함수를 매개변수로 받아들이거나 반환값으로 함수를 사용하는 함수를 고차 함수(high-order function)라고 함
  - 호출자는 무엇(what)이 되어야 하는지 만을 지정
  - 데이터를 실제 처리하는 방법(how)은 고차 함수에서 처리
  - MapReduce 컴퓨팅 패러다임에서 map(), reduce() 함수는 고차 함수

일급 함수 – 함수 타입과 값

- 함수의 타입은 함수의 입력 타입과 반환값 타입으로 표시

```
([<type>, ...]) => <type>
```

```
scala> def double(x: Int): Int = x * 2
double: (x: Int)Int

scala> double(5)
res0: Int = 10

scala> val myDouble: (Int) => Int = double
myDouble: Int => Int = <function1>

scala> myDouble(5)
res1: Int = 10

scala> val myDoubleCopy = myDouble
myDoubleCopy: Int => Int = <function1>

scala> myDoubleCopy(5)
res2: Int = 10
```

```
scala> def double(x: Int): Int = x * 2
double: (x: Int)Int

scala> val myDouble = double _
myDouble: Int => Int = <function1>

scala> val amount = myDouble(20)
amount: Int = 40
```

```
scala> def max(a: Int, b: Int) = if (a > b) a else b
max: (a: Int, b: Int)Int

scala> val maximize: (Int, Int) => Int = max
maximize: (Int, Int) => Int = <function2>

scala> maximize(50, 30)
res3: Int = 50
```

일급 함수 – 고차 함수

Higher-Order Functions

```
scala> def safeStringOp(s: String, f: String => String) = {
     |   if (s != null) f(s) else s
     | }
safeStringOp: (s: String, f: String => String)String

scala> def reverser(s: String) = s.reverse
reverser: (s: String)String

scala> safeStringOp(null, reverser)
res4: String = null

scala> safeStringOp("Ready", reverser)
res5: String = ydaeR
```


일급 함수 – 익명 함수 (Anonymous Function)

- 익명 함수 (anonymous function)은 이름 없는 함수
  - 함수 리터럴 (Function Literal) 또는 람다 표현식 (Lambda Expression) 이라고도 함

```
([<identifier>: <type>, ... ]) => <expression>
```

```
scala> val doubler = (x: Int) => x * 2
doubler: Int => Int = <function1>

scala> val doubled = doubler(22)
doubled: Int = 44
```

```
scala> val greeter = (name: String) => s"Hello, $name"
greeter: String => String = <function1>

scala> val hi = greeter("World")
hi: String = Hello, World
```

```
scala> def max(a: Int, b: Int) = if (a > b) a else b
max: (a: Int, b: Int)Int

scala> val maximize: (Int, Int) => Int = max
maximize: (Int, Int) => Int = <function2>

scala> val maximize = (a: Int, b: Int) => if (a > b) a else b
maximize: (Int, Int) => Int = <function2>

scala> maximize(84, 96)
res6: Int = 96
```

함수 정의

함수를 값으로 저장

익명 함수로 재정의

일급 함수 – 부분 적용 함수

❑ 부분 적용 함수 (Partially-Applied Function)

- 매개변수의 일부만 적용된 함수
- 적용되지 않은 인자는, 나중에 다시 받음
- 미리 적용할 인자는 값을 주고, 다른 인자는 _ (underscore) 표시
  - 스칼라에서 _ (underscore)는 문맥에 따라 여러 의미를 가짐

```
scala> def factorOf(x: Int, y: Int) = y % x == 0
factorOf: (x: Int, y: Int)Boolean
```

```
scala> val f = factorOf _
f: (Int, Int) => Boolean = <function2>

scala> val x = f(7, 20)
x: Boolean = false
```

```
scala> val multipleOf3 = factorOf(3, _: Int)
multipleOf3: Int => Boolean = <function1>

scala> val y = multipleOf3(78)
y: Boolean = true
```


일급 함수 – 커링 (Currying)

❑ 커링 (Currying)은 여러 개의 매개변수 그룹들로 분리하여 부분 적용 함수를 더 명확히 표현

```
scala> def factorOf(x: Int)(y: Int) = y % x == 0
factorOf: (x: Int)(y: Int)Boolean

scala> val isEven = factorOf(2) _
isEven: Int => Boolean = <function1>

scala> val z = isEven(32)
z: Boolean = true
```

4. 클래스

클래스 정의와 인스턴스

- 클래스 정의하고 인스턴스를 생성
  - *클래스이름@16진수 문자열* 출력
    - 인스턴스의 JVM의 내부 참조 표시
    - 16진수 문자열은 JVM의 java.lang.Object.toString 메서드가 출력

```
scala> class User
defined class User

scala> val u = new User
u: User = User@7a8c8dcf
```

- 클래스 안에서 메서드는 def로, 필드는 val로 정의
  - 메서드는 단지 클래스(객체)의 상태를 접근하는 함수
  - override 키워드로 toString 메서드를 재정의 예

```
scala> class User {
     |   val name: String = "Yubaba"
     |   def greet: String = s"Hello from $name"
     |   override def toString = s"User($name)"
     | }
defined class User

scala> val u = new User
u: User = User(Yubaba)

scala> println( u.greet )
Hello from Yubaba
```

생성자 (constructor)

- 스칼라에서는 생성자가 특별한 메서드로 따로 존재하지 않음
  - 클래스 몸체에서 메서드 정의 부분 밖에 있는 모든 코드가 생성자
  - 예
    - color 값은 if/else 식에 의해 초기화
    - 스칼라는 대부분의 구성 요소가 문(statement, 반환 값이 없는 문장)이 아니고 식(expression, 결과를 반환하는 문장)이라는 점에서 식 중심의 언어

```
class Calculator(brand: String) {
  /**
   * 생성자
   */
  val color: String = if (brand == "TI") {
    "blue"
  } else if (brand == "HP") {
    "black"
  } else {
    "white"
  }

  // 인스턴스 메소드
  def add(m: Int, n: Int): Int = m + n
}
```


클래스 상속

- 스칼라에서는 extends 키워드를 사용하여 다른 클래스를 확장 (상속)

```
scala> class A {
     |   def hi = "Hello from A"
     |   override def toString = getClass.getName
     | }
defined class A

scala> class B extends A
defined class B

scala> class C extends B { override def hi = "hi C -> " + super.hi }
defined class C

scala> val hiA = new A().hi
hiA: String = Hello from A

scala> val hiB = new B().hi
hiB: String = Hello from A

scala> val hiC = new C().hi
hiC: String = hi C -> Hello from A
```

추상 클래스

- 추상 클래스(abstract class)는 메서드 정의는 있지만 구현은 없는 클래스
  - 이를 상속한 하위클래스에서 메서드를 구현
  - 추상 클래스의 인스턴스를 만들 수는 없음

```
scala> abstract class Car {
     |   val year: Int
     |   val automatic: Boolean = true
     |   def color: String
     | }
defined class Car

scala> new Car()
<console>:9: error: class Car is abstract; cannot be instantiated
              new Car()
```

```
scala> class RedMini(val year: Int) extends Car {
     |   def color = "Red"
     | }
defined class RedMini

scala> val m: Car = new RedMini(2005)
m: Car = RedMini@5f5a33ed
```

```
scala> class Mini(val year: Int, val color: String) extends Car
defined class Mini

scala> val redMini: Car = new Mini(2005, "Red")
redMini: Car = Mini@1f4dd016

scala> println(s"Got a ${redMini.color} Mini")
Got a Red Mini
```


apply 메서드

- apply 메서드
  - 이름이 없이 호출되는 메서드
  - 메서드 이름 없이 괄호 사용하여 적용

```
scala> class Multiplier(factor: Int) {
     |   def apply(input: Int) = input * factor
     | }
defined class Multiplier

scala> val tripleMe = new Multiplier(3)
tripleMe: Multiplier = Multiplier@339cde4b

scala> val tripled = tripleMe.apply(10)
tripled: Int = 30

scala> val tripled2 = tripleMe(10)
tripled2: Int = 30
```

5. 객체, 케이스 클래스, 트레이트

객체 (object)

- 객체는 object 키워드로 선언
  - 클래스와 유사하나, 1개의 인스턴스만 생성
    - 객체의 매개변수 기술할 수 없음
  - 객체는 최초의 접근 시 JVM에서 자동으로 인스턴스화 됨
    - 처음 접근(호출) 시에만 생성자 호출

```
scala> object Hello { println("in Hello"); def hi = "hi" }
defined object Hello

scala> println(Hello.hi)
in Hello
hi

scala> println(Hello.hi)
hi
```

객체 – 순수 함수

- 객체의 메서드로는 순수 함수와 외부 입출력을 이용하는 함수에 적합
  - 주어진 입력 값으로만 계산하여 부작용이 없고 참조에 투명

```
scala> object HtmlUtils {
     |   def removeMarkup(input: String) = {
     |     input
     |       .replaceAll("""</?\w[^>]*>""","")
     |       .replaceAll("<.*>","")
     |   }
     | }
defined object HtmlUtils

scala> val html = "<html><body><h1>Introduction</h1></body></html>"
html: String = <html><body><h1>Introduction</h1></body></html>

scala> val text = HtmlUtils.removeMarkup(html)
text: String = Introduction
```


객체 – main 메서드

- 객체에 main 메서드를 사용하여 애플리케이션의 진입점 기술
  - scalac 명령으로 컴파일
  - scala 명령으로 실행
    - CLASSPATH 변수가 등록되어있는 경우 스칼라는 기본적으로 현재 디렉토리를 클래스패스로 등록하지 않기 때문에 실행할 때 현재 디렉토리를 클래스패스로 명시적으로 추가
      $ scala -cp . Date
  - *윈도우인 경우 스칼라 바이너리 설치 필요*
    - *https://downloads.lightbend.com/scala/2.12.4/scala-2.12.4.msi*

```
$ cat > Date.scala
object Date {
  def main(args: Array[String]) {
    println(new java.util.Date)
  }
}

$ scalac Date.scala

$ scala Date
Mon Sep 01 22:03:09 PDT 2014
```

케이스 클래스 (case class)

- 케이스 클래스 (case class)는 손쉽게 내용을 어떤 클래스에 저장하고, 그에 따라 매치(비교)를 하고 싶은 경우 사용
  - 클래스 정의 앞에 case 키워드로 생성
  - new를 사용하지 않고도 케이스 클래스의 인스턴스 생성이 가능
  - apply, copy, equals, …. 등의 메서드 제공

```
scala> case class Character(name: String, isThief: Boolean)
defined class Character

scala> val h = Character("Hadrian", true)
h: Character = Character(Hadrian,true)

scala> val r = h.copy(name = "Royce")
r: Character = Character(Royce,true)

scala> h == r
res0: Boolean = false

scala> h match {
     |   case Character(x, true) => s"$x is a thief"
     |   case Character(x, false) => s"$x is not a thief"
     | }
res1: String = Hadrian is a thief
```


트레이트 (trait) (1)

- 스칼라의 트레이트(trait)
  - 다중 상속이 가능한 클래스 유형
    - with 키워드로 여러 개의 트레이트를 확장 (상속)
  - 자바 interface + 추상 클래스
  - 객체와 마찬가지로 매개변수 기술할 수 없음

```
scala> trait HtmlUtils {
     |   def removeMarkup(input: String) = {
     |     input
     |       .replaceAll("""</?\w[^>]*>""","")
     |       .replaceAll("<.*>","")
     |   }
     | }
defined trait HtmlUtils

scala> class Page(val s: String) extends HtmlUtils {
     |   def asPlainText = removeMarkup(s)
     | }
defined class Page

scala> new Page("<html><body><h1>Introduction</h1></body></html>").asPlainText
res2: String = Introduction
```

트레이트 (2)

```
scala> trait Base { override def toString = "Base" }
defined trait Base

scala> class A extends Base { override def toString = "A->" + super.toString }
defined class A

scala> trait B extends Base { override def toString = "B->" + super.toString }
defined trait B

scala> trait C extends Base { override def toString = "C->" + super.toString }
defined trait C

scala> class D extends A with B with C { override def toString = "D->" +
  super.toString }
defined class D

scala> new D()
res50: D = D->C->B->A->Base
```


6. 컬렉션

컬렉션 (collection)

❑ 컬렉션 (collection)

- 배열, 리스트, 맵, 집합과 같이 특정 타입의 하나 이상의 값을 갖는 자료 구조
- 스칼라는 자바와 같이 고성능의, 객체지향적인, 타입-매개변수화된 컬렉션 프레임워크를 제공
- 변경 불가능 (immutable), 변경 가능 (mutable) 2가지 버전
  - 특별한 상황이 아니면, 변경 불가능 버전을 사용함
- 데이터를 반복하고 처리하는 범용 메서드(고차함수) 제공
- 자동으로 import되는 컬렉션
  - List – 리스트
  - Set – 집합
  - Map – 맵 , 파이썬의 딕셔너리(Dictionary)
  - Array – 배열
  - *Seq – 순서가 있는 자료형 (리스트, 배열 등…)*
  - *Vector*
  - *Tuple – 서로 다른 여러 자료형을 동시에 가짐*

리스트 (List)

❑ 리스트 (List)는 단방향 연결 리스트

```
scala> val numbers = List(32, 95, 24, 21, 17)
numbers: List[Int] = List(32, 95, 24, 21, 17)

scala> val colors = List("red", "green", "blue")
colors: List[String] = List(red, green, blue)

scala> println(s"I have ${colors.size} colors: $colors")
I have 3 colors: List(red, green, blue)
```

```
scala> val colors = List("red", "green", "blue")
colors: List[String] = List(red, green, blue)

scala> colors.head
res0: String = red

scala> colors.tail
res1: List[String] = List(green, blue)

scala> colors(1)
res2: String = green

scala> colors(2)
res3: String = blue
```

```
scala> val numbers = List(32, 95, 24, 21, 17)
numbers: List[Int] = List(32, 95, 24, 21, 17)

scala> var total = 0; for (i <- numbers) { total += i }
total: Int = 189

scala> val colors = List("red", "green", "blue")
colors: List[String] = List(red, green, blue)

scala> for (c <- colors) { println(c) }
red
green
blue
```

집합 (Set)과 맵 (Map)

- 집합 (Set)은 리스트와 유사하고 원소의 중복이 허용 안됨
- 맵 (Map)은 키-값 (key-value) 저장소로 파이썬의 딕셔너리 (해시)에 해당

```
scala> val unique = Set(10, 20, 30, 20, 20, 10)
unique: scala.collection.immutable.Set[Int] = Set(10, 20, 30)
```

```
scala> val colorMap = Map("red" -> 0xFF0000, "green" -> 0xFF00,
  "blue" -> 0xFF)
colorMap: scala.collection.immutable.Map[String,Int] =
  Map(red -> 16711680, green -> 65280, blue -> 255)

scala> val redRGB = colorMap("red")
redRGB: Int = 16711680

scala> val cyanRGB = colorMap("green") | colorMap("blue")
cyanRGB: Int = 65535

scala> val hasWhite = colorMap.contains("white")
hasWhite: Boolean = false

scala> for (pairs <- colorMap) { println(pairs) }
(red,16711680)
(green,65280)
(blue,255)
```


배열 (Array)과 튜플 (Tuple)

- 배열 (Array)는 고정된 크기의 인덱스를 갖는 컬렉션

```
scala> val colors = Array("red", "green", "blue")
colors: Array[String] = Array(red, green, blue)

scala> colors(0) = "purple"  (1)

scala> colors (2)
res0: Array[String] = Array(purple, green, blue)
```

- 튜플 (Tuple)
  - 튜플은 순서가 있는 서로 다른 타입의 2개 이상의 값을 가짐
  - 리스트, 배열과 달리 튜플의 요소를 반복할 수 없음

```
scala> val info = (5, "Korben", true)
info: (Int, String, Boolean) = (5,Korben,true)
```

```
scala> val name = info._2
name: String = Korben
```

```
scala> val red = "red" -> "0xff0000"
red: (String, String) = (red,0xff0000)

scala> val reversed = red._2 -> red._1
reversed: (String, String) = (0xff0000,red)
```

컬렉션 고차함수 (1)

- 스칼라 컬렉션은 반복(iterate), 변환(map), 축소(reduce), 필터(filter) 등의 연산을 수행하는 고차함수의 메서드를 제공
  - *고차함수는 함수를 매개변수로 받아들이는 함수*
  - foreach(), map(), reduce()

```
scala> val colors = List("red", "green", "blue")
colors: List[String] = List(red, green, blue)

scala> colors.foreach( (c: String) => println(c) )
red
green
blue
```

```
scala> val sizes = colors.map( (c: String) => c.size )
sizes: List[Int] = List(3, 5, 4)
```

```
scala> val numbers = List(32, 95, 24, 21, 17)
numbers: List[Int] = List(32, 95, 24, 21, 17)
```

```
scala> val total = numbers.reduce( (a: Int, b: Int) => a + b )
total: Int = 189
```


컬렉션 고차함수 (2)

- filter(), flatten, flatMap()

```
scala> val numbers = List(1, 2, 3, 4)
numbers: List[Int] = List(1, 2, 3, 4)
```

```
scala> numbers.filter((i: Int) => i % 2 == 0)
res0: List[Int] = List(2, 4)
```

```
scala> List(List(1, 2), List(3, 4)).flatten
res0: List[Int] = List(1, 2, 3, 4)
```

```
scala> def isEven(i: Int): Boolean = i % 2 == 0
isEven: (i: Int)Boolean

scala> numbers.filter(isEven _)
res2: List[Int] = List(2, 4)
```

```
scala> val nestedNumbers = List(List(1, 2), List(3, 4))
nestedNumbers: List[List[Int]] = List(List(1, 2), List(3, 4))

scala> nestedNumbers.flatMap(x => x.map(_ * 2))
res0: List[Int] = List(2, 4, 6, 8)
```

컬렉션 고차함수 (3)

```
// 각 원소에 +1
scala> List(1, 2, 3) map (_ + 1)
result: List[Int] = List[2, 3, 4]

scala> val words = List("the", "quick", "brown", "fox")
result: words: List[java.lang.String] = List(the, quick, brown, fox)

// 각 원소의 길이
scala> words map (_.length)
result: List[Int] = List(3, 5, 5, 3)

// 각 역순 문자열
scala> words map (_.toList.reverse.mkString)
result: List[String] = List(eht, kciuq, nworb, xof)

// 각 원소를 List로 변환
scala> words map (_.toList)
result: List[List[Char]] = List(List(t, h, e), List(q, u, i, c, k), List(b, r, o, w, n), List

// flatten처럼 하나의 List로 변환
scala> words flatMap (_.toList)
result: List[Char] = List(t, h, e, q, u, i, c, k, b, r, o, w, n, f, o, x)

// foreach
scala> var sum = 0
result: sum: Int = 0

scala> List(1, 2, 3, 4, 5) foreach (sum += _)

scala> sum
result: In = 15 // 1+2+3+4+5 = 15
```


컬렉션 고차함수 (4)

- 리스트 걸러내기: `filter`, `partition`, `find`, `takeWhile`, `dropWhile`, `span`

```
// filter : 주어진 조건에 맞는 리스트만 반환
scala> List(1, 2, 3, 4, 5) filter (_ % 2 == 0)
result: List[Int] = List(2, 4)

scala> var words = List("the", "quick", "brown", "fox")
scala> words filter (_.length == 3)
result: List[String] = List(the, fox)

// partition : 동일한 원소를 가진 리스트와 필터로 제거된 원소들만 가진 리스트 2개 반환
scala> List(1, 2, 3, 4, 5) partition (_ % 2 == 0)
result: (List[Int], List[Int]) = (List(2, 4),List(1, 3, 5))

// find : filter와 동일하지만 함수 결과가 true인 첫 번째 원소만 찾음 Some(x) ???
scala> List(1, 2, 3, 4, 5) find (_ % 2 == 0)
result: Option[Int] = Some(2)

// find의 결과가 없는 경우 None 반환
scala> List(1, 2, 3, 4, 5) find (_ <= 0)
result: Option[Int] = None

// takeWhile : 피연산자로 받은 조건에 대해 첫 실패가 되기 전까지의 리스트 반환
scala> List(1, 2, 3, -4, 5) takeWhile (_ > 0)
result: List[Int] = List(1, 2, 3)

// dropWhile : takeWhile과 반대로 첫 조건이 실패가 나오기 전까지만 모두 제거 후 나머지 반환
scala> words dropWhile (_ startsWith "t")
result: List[String] = List(quick, brown, fox)

// span : takeWhile과 dropWhile을 가지는 순서쌍(splitAt 참고)
List(1, 2, 3, -4, 5) span (_ > 0)
result: (List[Int], List[Int]) = (List(1, 2, 3),List(-4, 5))
```

```
scala> List(1,2,3,4,5).filter( _ % 2 == 0)
res1: List[Int] = List(2, 4)
```

과 제

- 강의 시간의 실습 내용을 정리하여 제출
  - 스칼라 예제 실행 및 결과 확인
- 의미 있는 임의의 스칼라 프로그램 작성
  - 컬렉션의 고차함수는 반드시 포함
  - 프로그램 설명
  - 프로그램 코드 및 실행 결과

참고 자료

- 스칼라
  - https://www.scala-lang.org/
- 러닝 스칼라, 제이슨 스와츠, 김정인, 강성용 옮김, 제이펍, 2017.
  - Learning Scala, Jason Swartz, 2017, O'Reilly
  - https://github.com/swartzrock/LearningScalaMaterials/blob/master/Sources/source_listings.asciidoc