Laboratorium 1.

1. **Zapoznanie się ze strukturą dokumentacji Scali (Scaladoc)**

    1. Otwieramy w przeglądarce stronę http://www.scala-lang.org/api/current/#package
    2. W lewym panelu klikamy w “display packages only” => kliknięty element zmienia się na “display all entities”
    3. Klikamy w “display all entities” => wracamy do stanu początkowego
    4. W górnej części lewego panelu, w polu wyszukiwania wpisujemy “math”
    5. Klikamy w “scala.math” i przeglądamy zawartość strony
    6. W głównym panelu klikamy w “Linear Supertypes”
    7. W głównym panelu klikamy w “Content Hierarchy”
    8. W głównym panelu klikamy w (Source) “package.scala” i przeglądamy zawartość pliku package.scala
    9. *[Wracamy do poprzedniej strony]* W głównym panelu, w polu wyszukiwania wpisujemy “abs”
    10. W lewym panelu klikamy w “BigInt”
    11. Klikamy kolejno w “Linear Supertypes”, “Type Hierarchy” i BigInt.scala
    12. W głównym panelu klikamy w elementy:
        - Inherited/ BigInt, Serializable (i kolejne w wierszu)
        - Visibility/ Public, All
        - Ordering/ Alphabetic, By inheritance
        - Implicitly/ by any2stringadd, by StringFormat (i kolejne w wierszu)
    13. *[Wracamy do poprzedniej strony]* W lewym panelu klikamy kolejno w “c”, “o” obok “BigInt” oraz “o” i “t” obok “Integral”
    14. Klikamy ponownie w BigInt
    15. W głównym panelu, w polu wyszukiwania wpisujemy “toByteArray”, rozwijamy opis (klikamy w trójkąt w lewej części wiersza)

        Zadanie:
        Znaleźć metodę sprawdzającą, czy dana liczba typu BigInt może być “bezstratnie przekonwertowana” na typ Int

2. **Uruchomienie interpretera Scali**

    1. Otwieramy okno konsoli/terminala
    2. Wpisujemy w konsoli “scala”

    ```
    $ scala
    Welcome to Scala version ...
    Type in expressions to have them evaluated.
    Type :help for more information.

    scala>
    ```

    3. Wpisujemy `:help` i analizujemy dostępne komendy (wszystkie zaczynają się dwukropkiem)
    4. Wpisujemy

    ```
    scala> println("Hello!")
    ```

    i naciskamy `[Enter]`

    5. Wpisujemy

    ```
    scala> 2 + 2
    ```

    i naciskamy `[Enter]`

    6. Wpisujemy

    ```
    scala> "Hello".
    ```

    naciskamy `[Tab]` i obserwujemy dostępne metody

    7. Dopisujemy “l” (jak “Lampa”)

    ```
    scala> "Hello".l
    ```

    naciskamy `[Tab]` i obserwujemy dostępne metody

    8. Dopisujemy `"e"`

    ```
    scala> "Hello".le
    ```

    naciskamy `[Tab]` -> `scala> "Hello".length`
    naciskamy `[Enter]`

    9. Naciskamy kilka razy klawisz [`Strzałka w górę` , potem `[Strzałka w dół]` (“historia”)
    10. Staramy się wpisać dwie linie:

    ```
    scala> println("Hello") [Enter]
    println("world!")
    ```

    11. Wpisujemy

    ```
    scala> :paste
    ```

    12. Wpisujemy

```
println("Hello")
println("world!")
```

13. Wpisujemy `[Ctrl+D]`
14. Wpisujemy

```
scala> 2 * (
3 +
4)
```

naciskamy `[Enter]`

15. Wpisujemy

```
scala> :paste
```

a następnie

```
val x = 2 +
3
```

`[Ctrl+D]`

16. Wpisujemy

```
scala> :paste
```

a następnie

```
val x = 2
+ 3
```

`[Ctrl+D]`

3. **Pierwszy skrypt w Scali**

1. Otwieramy nowe okno konsoli/terminala
2. Tworzymy nowy plik, nadajemy mu nazwę hello.scala
3. Wpisujemy w pliku (dodajemy jedną linię)

```
println("Hasta la vista, baby")
```

i zapisujemy plik.

4. W oknie konsoli/terminala wpisujemy

```
$ scala hello.scala
```

naciskamy `[Enter]`

5. *(W tym samym oknie)* Uruchamiamy interpreter Scali

```
$ scala
```

naciskamy `[Enter]`

6. Wpisujemy

```
scala> :load hello.scala
```

naciskamy `[Enter]`

4. **Zmienne i inferencja typów**

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy [Enter]

```
scala> var x1: Int = 3
scala> x1
scala> x1 = 5
scala> var x2 = 10.3
scala> x2
scala> val c1: Int = 11
scala> c1
scala> val c2 = 45.1
scala> c2 = 32.3
scala> val c2 = 32.3
scala> val źdźbło = 3
scala> val `pięć źdźbeł` = 5 * źdźbło
scala> val ~^~ = 4
scala> val |-<*>-| = 4
scala> val x1 = 4, x2 = 5
scala> val x1 = 4; x2 = 5
scala> val x1 = 4; val x2 = 5
scala> val x3 = x2++
scala> val x3 = ++x2
scala> val x3 = x2 += 1
scala> val x4 = _
scala> val x4: Int = _
scala> var x4 = _
scala> var x4: Int = _
```

2. Wpisujemy

```
scala> c2.
```

naciskamy `[Tab]` i wybieramy

```
scala> c2.toInt
```

5. **Instrukcje vs. wyrażenia**

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy [Enter]

```
scala> var x = 0
scala> val y = 5.4
scala> val sum = x + y // Jaki jest typ sum?
scala> val xr1 = (x = 2) // Jaki jest typ xr1 (co to oznacza?)
scala> val xr2 = x = 2
scala> val xr3 = (x == 0) // Jaki jest typ xr2 (co to oznacza?)
scala> val xr4 = (x += 2) // Jaki jest typ xr3 (co to oznacza?)
scala> x = x + 2 // Porównaj wynik z poprzednim
```

2. Wpisujemy poniższe bloki i na końcu każdego naciskamy [Enter]

```
scala> { val x = 10; 2 * x + 1 } // Jaka jest "wartość bloku"

scala> val b1 = {
println("The answer to the Ultimate Question ")
42
}

scala> val b2 = {
val answer = 42;
println("The answer to the Ultimate Question is " + answer)
}
```

Dlaczego b1 != b2 ?

6. **Wyrażenie warunkowe (`if-else`)**

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> val i = -1
scala> val abs_i = if (i >= 0) i else -i
scala> val ifNoElseRes = if (i >= 0 || i % 2 == 0) println("Nonn
egative or even") // Jaki jest typ ifNoElseRes
```

2. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> val i = 1
scala> val ifRes = if (i > 0) i else 0
scala> val halfIfRes1 = if (i > 10) i
scala> val halfIfRes2 = if (i > 10) i else ()
```

Porównaj typy wartości `ifRes` , `halfIfRes1` i `halfIfRes2` .

7. **Pętla `while`**

1. Wpisujemy poniższy kod (`:paste` + ... + `[ctrl-d]`)

```
var i = 0
while (i < 5) {
println(i)
i += 1
}
```

Zadanie:
Wpisać poniższy kod do pliku while.scala i uruchomić go z konsoli/terminala oraz w REPL (`:load`)

2. Wpisujemy poniższy kod (: `paste` + ... + `[ctrl-d]`)

```
var i = 10
val whileRes = while (i >= 0) {
 println(i)
 i -= 2
 true
}
```

Jaki jest typ `whileRes` ?

8. **Pętla `do-while`**

1. Wpisujemy poniższy kod w pliku `whiledo.scala` i uruchamiamy go z konsoli w REPL (`:load`)

```
var i = 0
do {
 if (i % 2 == 0) println(i)
 i += 1
} while (i < 10)
```

Zadanie:

Zmienić w pliku whiledo.scala pętlę “ `do-while` “ na “ `while` “ i uruchomić skrypt

1. Wpisujemy poniższy kod (`:paste` + ... + `[ctrl-d]`)

```
var i = 0
val doWhileRes = do {println(i); i += 1; true} while (i < 4
)
```

Jaki jest typ `doWhileRes` ?

9. **Pętla** `for`

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> for (i <- 1 to 3) println(i)
scala> for (i <- 1 until 3) println(i)
scala> val expr1 = 1 to 3
scala> val expr2 = 1 until 3
scala> for (i <- 1.to(3)) println(i)
scala> for (i <- 1.until(3)) println(i)
scala> val expr1 = 1.to(3)
scala> val expr2 = 1.until(3)
```

2. Wpisujemy poniższą linie i naciskamy `[Enter]`

```
scala> for (i <- 1 to 10; x = 2 * i + 1; if (x % 3 == 0)) printl
n(i, x)
scala> val forLoopRes = for (i <- 1 to 10; x = 2 * i + 1; if (x
% 3 == 0)) {println(i, x); i}
```

Jaki jest typ `forLoopRes` ?

Zadanie:
Zmienić pętle “ `for` “

```
for (i <- 1 to 10; x = 2 * i + 1; if (x % 3 == 0)) println(i, x)
```

na pętlę “ `while` “. Skrypt wpisać w pliku `for2while1.scala`

3. Wpisujemy poniższą linie i naciskamy `[Enter]`

```
scala> for (i <- 1 to 3; j <- 1 to 4) println(i,j)
```

Zadanie:
Napisać pętlę “ `for` “ wypisującą indeksy dwuwymiarowej macierzy o wymiarach 5 x 5 leżące ponad główną przekątną

10. **Wyrażenie** `for-yield`

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> for (i <- 1 to 3) yield(i)
scala> val xsv = for (i <- 1 to 3) yield(2 * i + 1)
```

2. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> val xsa = for (i <- (1 to 3).toArray) yield(i) // def toA
rray: Array[A]
scala> val xsl = for (i <- (1 to 3).toList) yield(i) // def toLi
st: scala.List[A]
scala> val xss1 = for (i <- (1 to 3).toSet) yield(2 * i) // def
toSet[B >: A]: Set[B]
scala> val xss2 = for (i:Int <- (1 to 3).toSet) yield(i * i)
scala> val xss3 = for (i <- (1 to 3).toSet[Int]) yield {if (i %
2 == 0) i * i else 2 * i + 1}
scala> val xss4 = for (i <- (1 to 3).toSet[Int]) yield {if (i %
2 != 0) i * i}
```

Jaki jest typ zwracany przez wyrażenie “ `for` “?

Zadanie (opcjonalne):
Przepisać kod wyrażenia

```
val xsa = for (i <- (1 to 3).toArray) yield(i)
```

na pętlę “ `while` “. Wskazówka:

```
scala> val xsa = new Array[Int](3)
xsa: Array[Int] = Array(0, 0, 0)
scala> xsa(0) = 0
scala> val xsa0 = xsa(0)
xsa0: Int = 0
`
```

11. **Metody vs. funkcje w Scali** (“pierwsza wzmianka”)

1. Definiujemy metodę “ `abs` “

```
scala> def abs(x: Double): Double = { if (x >= 0) x else -x }
abs: (x: Double)Double
```

2. Obliczamy wartość `abs(-5)`

```
scala> val absMinus5 = abs(-5)
absMinus5: Double = 5.0
```

3. *(Materiał opcjonalny)* Definiujemy funkcję " `abs` "

```
val absAsFunction: (Double) => Double = abs
absAsFunction: Double => Double = <function1>
```

lub inny sposób:

```
scala> val absAsFunction = abs _
absAsFunction: Double => Double = <function1>
```

4. *(Materiał opcjonalny)* Obliczamy `abs(-3)` wykorzystując funkcję `absAsFunction`

```
scala> val x1 = absAsFunction(-3)
scala> val x2 = absAsFunction.apply(-3) // równoważne powyższej
linii
x: Double = 3.0
```

5. *(Materiał opcjonalny)* Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> val absToString = abs.toString
scala> val absFunToString = absAsFunction.toString
```

Uwaga: funkcje będą omówione podczas zajęć dotyczących programowania funkcyjnego

12. **Typ rezultatu metody**

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> def abs(x: Double): Double = if (x >= 0) x else -x
scala> def abs(x: Double) = if (x >= 0) x else -x
scala> def factorial (n: Int) = if (n <= 0) 1 else n * factorial
(n-1)
scala> def factorial (n: Int): Int = if (n <= 0) 1 else n * fact
orial(n-1)
scala> def f1(x: Int) = if (x > 0) 1 else 2.3 //Jaki jest typ wy
niku?
scala> def f2(x: Int) = if (x >= 0) 1 else "x is negative" //Jak
i jest typ wyniku?
```

2. Obliczamy `f1(3)` , `f1(-10)` , `f2(100)` , `f2(-45)`

Zadanie:
Zdefiniować jeszcze raz metody `f1` i `f2` , podając jawnie typ rezultatu

3. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> def printlnArg(a: Int): Unit = { println(a) }
scala> def printlnArg(a: Int) = println(a)
scala> def printlnArg(a: Int) { println(a) }
scala> def printlnArg(a: Int) println(a)
```

Zadanie:
Porównaj wyniki (sygnatury metod) powyższych definicji

13. **Funkcje o zmiennej liczbie parametrów** (*variadic functions*)

1. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> def printAll(args: Int*) {for (arg <- args) println(arg)}
scala> printAll(1,2,3,4,6)
scala> printAll(1,10,100,1000,10000)
```

2. Wpisujemy pojedynczo poniższe linie i na końcu każdej naciskamy `[Enter]`

```
scala> def f1(head: Int, tail: Int*) = {println(head); for (arg
<- tail) println(arg)}
scala> f1(1,2,3,4,5)
scala> f1(1, (2 to 20): _*)
```

3. Wpisujemy pojedynczo poniższe fragmenty kodu i na końcu każdego naciskamy `[Enter]`

```
scala> def f2(ints: Int*, theLast: Int) = {
 for (arg <- ints) println(arg); println(theLast)
}

scala> def f3(ints: Int*, doubles: Double*) = {
 for (arg <- ints) println(arg)
 for (arg <- doubles) println(arg)
}
```

Przeanalizuj opisy błędów.

4. Wpisujemy pojedynczo poniższe linie, na końcu każdej naciskamy [Enter]

```
scala> def f4(args: Any*) = for (arg <- args) println(arg)
scala> f4(1, 3.14, true, "Jak dobrze być pisanką!")
```

14. **Wartości domyślne parametrów**

    - Wpisujemy pojedynczo poniższe fragmenty kodu i na końcu każdego naciskamy `[Enter]`

    ```
    scala> def printName(name: String = "John", surname: String = "D
    oe") {
      println(name + " " + surname)
    }

    scala> printName()
    scala> printName("John")
    ```

15. **Argumenty nazwane (vs. pozycyjne)**

    Wpisujemy pojedynczo poniższe fragmenty kodu i na końcu każdego naciskamy `[Enter]`

    ```
    scala> printName(surname = "Smith")
    scala> printName(name = "Andrew")
    scala> printName(surname = "Black", name = "Joe")
    scala> printName("Black", "Joe") // Wyjaśnij różnicę względem poprze
    dniego wywołania
    ```

16. **Metody bezparametrowe**

    Wpisujemy pojedynczo poniższe fragmenty kodu i na końcu każdego naciskamy `[Enter]`

    ```
    scala> def f4() = 2 * 2
    scala> f4()
    scala> f4
    scala> def f5 = 2 * 2
    scala> f5
    scala> f5()
    ```

    Przeanalizuj “zachowanie” obu metod.