Chapter 1

Expressions and Messages

This exercise is about reading and understanding Pharo expressions, and differentiating between different types of messages and receivers. Note that in the expressions you will be asked to read and executed, you can assume that the implementation of methods generally corresponds to what their message names imply (i.e., `2 + 2 = 4`). Note that most of the expressions we use in the exercises are expressions that you can execute in Pharo, so do not hesitate. Some of the expressions explained in https://medium.com/concerning-pharo/elegant-pharo-code-bb590f0856d0

1.1 Exercises

For each of the k expressions below, fill in the answers:

- What is the receiver object?
- What is the message selector?
- What is/are the argument (s)?
- What is the message?
- What is the result returned by evaluating this expression?

```
3 + 4
```

```
Date today
```

```
anArray at: 1 put: 'hello'
anArray at: i
```

```
#(2 33 −4 67) collect: [ :each | each abs ]
```

```
25 @ 50
```

```
SmallInteger maxVal
```

```
#(a b c d e f) includesAll: #(f d b)
```

Exercise: Literal objects

What kind of object does the following literal expressions refer to?

```
'Hello, Dave'
```

```
#Node1
```

```
#(2 33 4)
```

```
[ :each | each scale: 1.5 ]
```

Exercise: Scope

- What can one assume about a variable named `Transcript`?
- What can one assume about a variable named `rectangle`?

Exercise:

Examine the following expressions. What is the value returned by the execution of the snippet?

```
| anArray |
anArray := #('first' 'second' 'third' 'fourth').
anArray at: 2.
```

Exercise: unneeded parentheses

Putting more parentheses than necessary is a good way to get started. Such practice however leads to less read

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

```
x between: (pt1 x) and: (pt2 y)
```

```
(x isZero)
   ifTrue: [....]
(x includes: y)
   ifTrue: [....]
```

```
(OrderedCollection new)
  add: 56;
  add: 33;
  yourself
```

```
(Integer primesUpTo: 64) sum
```

```
('http://www.pharo.org' asUrl) retrieveContents
```

```
('2014-07-01' asDate - '2013/2/1' asDate) days
```

```
(((ZnEasy getPng: 'http://pharo.org/web/files/pharo.png')
 asMorph) openInWindow)
```

```
((#(a b c d e f) asSet) intersection: (#(f d b) asSet))
```

1.2 Exercise: Results

Guess what are the results of the following expressions:

```
6 + 4 / 2
```

```
1 + 3 negated
```

```
1 + (3 negated)
```

```
2 raisedTo: 3 + 2
```

```
2 negated raisedTo: 3 + 2
```

```
#(a b c d e f) includesAll: #(f d b)
```

Exercise: Execution sequence

Examine each of the following expression and write down the sequence of steps of their execution (which message is executed first and so on)

```
Date today daysInMonth
```

```
5@5 extent: 6.2 truncated @ 7
```

```
Transcript show: (45 + 9) printString
```

```
('2014-07-01' asDate - '2013/2/1' asDate) days
```

```
42 factorial decimalDigitLength
```

```
(ZnServer startDefaultOn: 8080)
   onRequestRespond: [ :request | ZnResponse ok: (ZnEntity with: DateAndTime
   now printString) ]
```

```
(1914 to: 1945) count: [ :each | Year isLeapYear: each ].
```

```
$/ join: ($- split: '1969-07-20') reverse
```

```
DateAndTime fromUnixTime:
   ((ByteArray readHexFrom: 'CAFEBABE4422334400FF')
     copyFrom: 5 to: 8) asInteger
```

```
(String new: 32) collect: [ :each | 'abcdef' atRandom ]
```

```
'http://www.pharo.org' asUrl saveContentsToFile: 'page.html'
```

```
'^.*.jpg' asRegex in: [ :regex |
  '/tmp/foo.txt' asFileReference contents lines
    select: [ :line | regex matches: line ] ]
```

Exercise: Examine the block expressions

```
| sum |
sum := 0.
#(21 23 53 66 87)do: [:item | sum := sum + item].
sum
```

What is the final result of sum? How could this piece of code be rewritten to use explicit array indexing (with the method at:) to access the array elements1? Test your version. Rewrite this code using `inject:into:`

Exercise: Comparing expressions

```
| array |
array := #(2 4 4 4 5 5 7 9).
((array − array average) squared sum / (array size − 1)) sqrt
```

```
[ :input | ((input − input average) squared sum / (input size − 1)) sqrt ]
   value: #(2 4 4 4 5 5 7 9)
```

```
#(2 4 4 4 5 5 7 9) in: [ :input |
  ((input − input average) squared sum / (input size − 1)) sqrt ]
```

```
#(2 4 4 4 5 5 7 9) stddev
```