```swift
/**
 Constants and Variables

 Exercise 1.1
 Declare a floating-point variable that represents temperature and assign the current temperature to it. If you
 don't have a thermometer, guess the temperature. :-)
 Then assign a different temperature to it. Pick one at random, it doesn't matter.
 */
var temperature: Float = 18.5
temperature = 25.8



/**
 Exercise 1.2
 Declare a constant integer value that represents the number of seconds in an hour and assign that number
 to it on the same line.
 Then try to assign a different value to the constant. Why doesn't it work?
 */
let secounds = 20
//secounds = 25
//Error because Cannot assign to value: 'secounds' is a 'let' constant

/**
 Exercise 1.3
 Declare an integer variable with an explicit type annotation and then one without.
 */
var a:Int
//var b //Error Type annotation missing in pattern

/**
 Exercise 1.4
 Declare a constant integer value that represents the number of wheels of a car. Don't assign it a value on the
 same line.
 On the next line, assign a value to the constant. Why does this work?
 */
let wheelsOfCar: Int
wheelsOfCar = 2 //It works because this is first time assign value to constant.

/**
 Exercise 1.5
 Declare the constant π using the name π, which you can type by pressing alt-p.
 */
let π = 3.14

/**
 Exercise 1.6
 Declare a variable using an emoji in the name.
 */
var 🌞 = "Sun"
```

```swift
/**
 Exercise 1.7
 Print a variable using the print() function.
 */
print(🌞)

/**
 Exercise 1.8
 What is the maximum value that can be stored in an Int16? Write the code that prints the maximum value of
 the Int16 type.
 */
print(Int16.max) // 32767

/**
 Exercise 1.9
 What type is the constant pi in the example below? Why? let pi = 3 + 0.141592654
 */
let pi = 3 + 0.141592654
print(type(of: pi)) //Double

/**
 Exercise 1.10
 What happens if you try the following code? Why? let myNumber: UInt = -17
 */
//let myNumber: UInt = -17 // Error Negative integer '-17' overflows when
 stored into unsigned type 'UInt'

/**
 Exercise 1.11
 What happens if you try the following code? Why? let bigNumber: Int16 = 32767 + 1
 */
//let bigNumber: Int16 = 32767 + 1 //Error Arithmetic operation '32767 + 1'
 (on type 'Int16') results in an overflow

/**
 Exercise 1.12
 Why does the following code not work? What do you need to add to it to make it work, if we absolutely want
 to store this value as an integer, i.e. 3, but we don't want to change the type of the variables?
let pi = 3.141592654
let approximatePi: Int = pi
 */
let pi2 = 3.141592654
let approximatePi: Int = Int(pi2)

/**
 Exercise 1.13 EXTRA CREDIT
 The following code will print true, which means that valueA and valueB are equal. Why are they equal? (The
 << is the bitshift left operator.)
 let valueA: Int16 = -0x8000
 let valueB: Int16 = 0x4000 << 1
 print(valueA == valueB)
 */
let valueA: Int16 = -0x8000
let valueB: Int16 = 0x4000 << 1
print(valueA == valueB) //because valueA = -32768 and valueB after bitshift =
 valueA.
/*
 Here's how bit shifting looks in Swift code:

 let shiftBits: UInt8 = 4   // 00000100 in binary
 shiftBits << 1             // 00001000
 shiftBits << 2             // 00010000
 shiftBits << 5             // 10000000
 shiftBits << 6             // 00000000
 shiftBits >> 2             // 00000001
 */
```

```
/**
```
Comments

Exercise 1.14
There are two types of comments. Single-line and multiline comments. Write one of both.
```
 */
//Single-line comment
/*
 Multiple line comment
 */
```

```
/**
```
Exercise 1.15
In Swift multiline comments can be nested. Write a nested multiline comment.
```
 */
/* This is the start of the first multiline comment.
/* This is the second, nested multiline comment. */
This is the end of the first multiline comment. */
```

```
/**
```
Semicolons
This is not really an exercise, but try to avoid using semicolons. If you have to use a semicolon, you are probably doing something wrong. :-) It's generally preferred code style to write each statement on its own line.
```
 */
//got it
```

```
/**
```
Tuples Exercise 1.16
Assign a tuple with two values in it to a constant named player. The values could represent the number of a hockey player and the name of the hockey player. For example, Igor Larionov whose number used to be 8.
```
 */
let player = (number: 8, name: "Igor Larionov")
print("Player number:", player.0)
print("Player name:", player.1)
```

```
/**
 Exercise 1.17
 OK, now you have a player tuple. Decompose (i.e. split) the number and the name into two
 constants named number and name. This could be done in at least three ways. Try all three. Optionals
 */
let number1 = player.0
let name1 = player.1

let number2 = player.number
let name2 = player.name

let (name3, number3) = player
print(name3, number3)

/**
 Exercise 1.18
 Can a constant have an optional type? If you're not sure, try it and see what happens.
 */
let tryConstantOptional: String?
tryConstantOptional = "Hello World"

/**
 Exercise 1.19
 Why doesn't this work? What needs to be added to value on the second line for this to work?
 let value: Int? = 17
 let banana: Int = value
 */
let value: Int? = 17
//let banana: Int = value //Error Value of optional type 'Int?' must be
 unwrapped to a value of type 'Int'
//Add default value
let banana: Int = value ?? 0
//or add force-unwrap sign !
let banana2: Int = value!

/**
 Exercise 1.20
 If value in the previous exercise had been nil, what would have happened if you force
 unwrapped value?
 */
//Error and App crash!!!!!!

/**
 Exercise 1.21
 What would be a better way to assign value to the banana constant?
 */
//Add default value or always use optional binding before use optional value.
```