1 Output

```
# evaluate expression and prints result.
irb> 4 + 5
=> 9
irb>
7 * 3
=> 21
irb> 'Hello, world!'
=> "Hello, world!"

# print text on standard output
irb> puts 'Hello, world!'
Hello, world!
=> nil
```

2 Dynamically typed variables

```
irb> v = nil        # Creates a new variable v on the stack holding
=> nil              # a reference to an instance of NilClass.

irb> v = 42         # Updates the v variable with a reference to an
=> 42               # a Fixnum object, with value 42.

irb> s = 'Hello'    # Creates a new variable s on the stack holding a
=> "Hello"          # reference to a String object with value 'Hello'.

irb> v + s          # Invokes the +-method on the 'Hello' String
                    # object referenced by the s variable. This will
                    # raise a TypeError as the String object does not
                    # know how to add a Fixnum to a String.

irb> v + s          # + is invoked on the Fixnum, resulting in a
                    # TypeError as String cannot automaticly be
                    # converted to Fixnum.

irb> v.to_s         # The to_s-method returns a printable string
=> "42"             # representation of the object. In this case
                    # "42".

irb> s + v.to_s     # Returns the concatenated string "Hello42"
=> "Hello42"

irb> s.methods      # Returns an array of the methods defined for the
=> ["upcase!",...]  # object referenced by s.
```

```
irb> s, v = v, s    # Multiple assignment: Swaps the values of v and s.
=> [42, "Hello"]
```

3 Lists

```
irb> l = [v, s]     # Creates a list(Array) containing 'Hello' and 42.
=> ["Hello", 42]

irb> l << 3.0       # All kinds of types may be mixed in a list.
=> ["Hello", 42, 3.0]

irb> l.pop          # Removes and returns the last element in list.
=> 3.0              # Can pop multiple values, i.e. l.pop(2) returns
                    # a list of the last two elements.

irb> r = 1..4       # Creates a new instance of the Range class
=> 1..4             # Useful when looping. x..y is inclusive the
                    # last element, x...y is exclusive.

irb> l = r.to_a     # Creates a new instance of the Range class
=> 1..4             # Useful when looping. x..y is inclusive the
                    # last element, x...y is exclusive.

irb> l = r.to_a     # Convert the Range to a list
=> [1, 2, 3, 4]

irb> l = l + [5, 6] # Concatenation of lists
=> [1, 2, 3, 4, 5, 6]

# Loop over each element in the list l and summarise in sum.
irb> l.each do |element|
irb*   the_sum += element
irb> end

# Above loop will crash as the_sum is unassigned, and therefore
# considered nil. NameError is raised as nil has no + method.

irb> the_sum = 0
irb> l.each do |element|
irb* the_sum += element
irb> end
=> [1,2,3,4,5,6] # iteration returns the iterated object
irb> puts the_sum
21
=> nil
```

```
# Alternate loop syntax
irb> l.each { |element| the_sum += element }
```

4 List access and slicing

```
irb> lst = ['a', 'b', 'c', 'd']
=> ["a", "b", "c", "d"]

# The index method [] takes one or two arguments.
# The first is which index to retrieve, negative index means
# count backwards from the end of the list. Second argument
# is how many consecutive elements to returns as a new list.

irb> lst[1]     # Returns 'b' as 1 is the second position in list.
=> "b"
irb> lst[-1]    # Returns the last element of the list.
=> "d"
irb> lst[7]     # Indexing outside of the list returns nil.
=> nil
irb> lst[1,1]   # Returns a new list containing "b".
=> ["b"]
irb> lst[0,3]   # Returns 3 elements from index 0.
=> ["a","b","c"]
irb> lst.dup    # Returns a copy of the list.
=> ["a", "b", "c", "d"]
irb> lst.reverse # Returns a copy of the list reversed.
["d", "c", "b", "a"]
```

5 Associative arrays (Hashes/Dictionaries/Maps?)

```
# Create a new Hash, with unordered mappings.
irb> h = { 'one'=>1, 'two'=>2, 'three'=>3, 'four'=>4 }
=> {"three"=>3, "two"=>2, "one"=>1, "four"=>4}
irb> h["one"]
1

# Update hash, key "three" refers the same value as
# key "four".
irb> h["three"] = h["four"]
=> 4

# Print string representation of the contents of the Hash.
irb> puts h.inspect
```

```
{"three"=>4, "two"=>2, "one"=>1, "four"=>4}

irb> h.keys
=> ["three", "two", "one", "four"]
irb> h.has_key? "four"
=> true
irb> h.has_value? 4 # Is this one of the values?
=> true

# Iterate over the key/value-pairs and print each.
irb> h.each { |key, value| puts "#{key} => #{value}" }
three => 4
two => 2
one => 1
four => 4
=> {"three"=>4, "two"=>2, "one"=>1, "four"=>4}
```