Question 1 (8 marks) The Collatz sequence starting at a positive integer n is generated by repeating the following rule until the value 1 is reached: if the current value is even, divide it by 2; if it is odd, replace it by $3 \cdot \text{value} + 1$. Implement the Julia function `collatzsteps(n)` which returns the number of steps needed to reach 1 starting from `n`. Use a `while` loop.

For example, the sequence for $n = 6$ is $6, 3, 10, 5, 16, 8, 4, 2, 1$, which takes 8 steps. A successful test for your implementation would be:

```
julia> using Test
julia> @test collatzsteps(6) == 8
Test Passed
julia> @test collatzsteps(1) == 0
Test Passed
```

Solution:

```
function collatzsteps(n)
  steps = 0
  while n != 1
    n = iseven(n) ? div(n, 2) : 3n + 1
    steps += 1
  end
  return steps
end
```

Question 2 Consider the following Julia code, run as a script:

```
x = 10

function g()
  x = 1
  for i in 1:3
    x += i
  end
  return x
end

println(g(), " ", x)
```

(a) What is printed?
Solution:
`7 10`. *The assignment* `x = 1` *inside* `g` *creates a new local variable which shadows the global* `x`. *The loop modifies this local variable* ($1 + 1 + 2 + 3 = 7$), *the global* `x` *remains* 10.

(b) Which one line would you insert into `g` so that the output becomes `7 7`? Where would you insert it?
Solution:
Insert `global x` *as the first line of the body of* `g` *(before* `x = 1`*). Now* `x` *in* `g` *refers to the global variable.*

Question 3 Consider the Julia code

```
A = [1 2 3; 4 5 6; 7 8 9]
B = A[2:3, [1,3]]
```

```
3 C = A[:, 2]
4 D = A .* A'
5 E = A[end, :]'
```

(a) What is the value of `B`?

(b) What is the value and the type of `C`?

(c) What is the value of `D[3,2]`?

Solution:
`B` *is the matrix* `[4 6; 7 9]`.
`C` *is the vector* `[2, 5, 8]` *of type* `Vector{Int64}` *(one-dimensional array).*
`D` *is the elementwise product of* `A` *with its transpose, so* `48`.
`E` *is the adjoint (row vector) of the last row* `[7, 8, 9]`*, so its size is* `(1, 3)`.

Question 4 (6 marks) Consider the Julia code

```
1 a = [1, 2, 3]
2 b = a
3 c = copy(a)
4 b[1] = 100
5 push!(c, 4)
6
7 f(v) = (v[end] = -1; v)
8 d = f(a)
```

What are the values of `a`, `b`, `c`, and `d` after running this code? What does the expression `d === a` evaluate to? Explain in one sentence. *Solution:*
`a = [100, 2, -1]`, `b = [100, 2, -1]`, `c = [1, 2, 3, 4]`, `d = [100, 2, -1]`, *and* `d === a` *is* `true`.
Arrays are stored by reference: `b = a` *and the return value of* `f` *refer to the same array in memory as* `a`*, whereas* `copy` *creates an independent array.*

Question 5 Let `s = "Julia is fun"`. State the result of each of the following expressions.

(a) `split(s)`

(b) `uppercase(s[1:5])`

(c) `replace(s, "u" => "U")`

(d) `"$(length(s)) chars: $(s[end-2:end])"`

Solution:
(a) `["Julia", "is", "fun"]` *(b)* `"JULIA"`
(c) `"JUlia is fUn"` *(d)* `"12 chars: fun"`

Question 6 (a) What does the following Unix command do? Explain the role of each of its two parts and of the symbol between them.

```
ls *.jl | tail
```

Solution:
`ls *.jl | tail` *lists all files in the current directory with the extension* `.jl`*. The pipe symbol* `|` *passes this output as input to* `tail`*, which only outputs the last 10 lines.*

Question 7 (10 marks)

(a) (6 marks) Consider the following sorting algorithm implemented as function `insertionsort!(v)` which sorts the vector `v` in ascending order in place using the insertion sort algorithm, and returns `v`.

```
function insertionsort!(v)
  for i in 2:length(v)
    key = v[i]
    j = i - 1
    while j >= 1 && v[j] > key
      v[j+1] = v[j]
      j -= 1
    end
    v[j+1] = key
  end
  return v
end

julia> @test insertionsort!([5, 2, 4, 6, 1, 3]) == [1, 2, 3, 4, 5, 6]
Test Passed
```

(b) (2 marks) What is the worst-case time complexity of this implementation in terms of the length n of `v`? Which input is a worst case?
Solution: $O(n^2)$. The worst case is a vector sorted in descending order: every new element has to be moved past all previously sorted elements ($1 + 2 + \cdots + (n-1)$ moves).

(c) (2 marks) What is the best-case time complexity and for which input does it occur?
Solution: $O(n)$ for an input which is already sorted: the inner `while` *loop stops immediately and every element is only compared once.*

Question 8 State the time complexity in Big-O notation of each of the following functions in terms of n (briefly justify your answer).

```
function g1(n)
  s = 0
  for i in 1:n, j in 1:i
    s += 1
  end
  return s
end

function g2(n)
  s = 0
  i = n
  while i >= 1
    s += 1
    i = div(i, 2)
  end
  return s
end
```

Solution:
`g1`*: $O(n^2)$. The body executes $\sum_{i=1}^{n} i = n(n+1)/2$ times.*
`g2`*: $O(\log n)$. The value of* `i` *is halved in every iteration, so there are about $\log_2 n$ iterations.*

Question 9 Consider the following results in the Julia REPL.

```
julia> typemax(Int8) + Int8(1)
-128

julia> 2^64
0

julia> 0.1 + 0.2 == 0.3
false
```

(a) Explain the first result.
Solution:
`typemax(Int8)` *is* 127*, which in binary is* `01111111`*. Adding one results in* `10000000`*, which in the two's complement representation of 8-bit signed integers is* -128*. Integer arithmetic in Julia silently wraps around (overflow) and does not throw an error.*

(b) Explain the second result and state how to compute 2^{64} correctly.
Solution:
Same reason: 2^{64} *does not fit into 64 bits, so the result wraps around to* 0*. Use arbitrary precision integers:* `big(2)^64` *gives* `18446744073709551616`*.*

(c) Explain the third result and state how to compare the two numbers in a meaningful way.
Solution:
0.1, 0.2*, and* 0.3 *cannot be represented exactly as binary floating point numbers; the representation and the addition have small rounding errors, so* `0.1 + 0.2` *differs from* `0.3` *in the last bit. Use* `isapprox(0.1 + 0.2, 0.3)` *which compares with a tolerance.*

Question 10 The derivative of a function f can be approximated by the forward difference

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

(a) Implement the Julia function `centraldiff(f, x, h)` which approximates $f'(x)$ using the central difference $\frac{f(x+h)-f(x-h)}{2h}$.
Solution:

```
centraldiff(f, x, h) = (f(x + h) - f(x - h)) / (2h)
```

(b) The absolute error of the forward difference for $f = \sin$ at $x = 1$ was computed for several values of h:

h	10^{-1}	10^{-4}	10^{-8}	10^{-12}	10^{-16}
error	$4.3 \cdot 10^{-2}$	$4.2 \cdot 10^{-5}$	$3.0 \cdot 10^{-9}$	$4.3 \cdot 10^{-5}$	$5.4 \cdot 10^{-1}$

Explain why the error does not keep decreasing as h becomes smaller.
Solution:
For large h *the truncation error of the approximation dominates (it decreases proportionally to* h*). For small* h*,* $f(x+h)$ *and* $f(x)$ *are almost equal and their difference suffers from cancellation (loss of significant digits) due to the finite precision of floating point numbers; dividing by the tiny* h *amplifies this rounding error. For* $h = 10^{-16} < \varepsilon_{\text{mach}}$ *one even obtains* $x + h = x$ *and the difference quotient is* 0*.*

Question 11 (10 marks)

(a) (7 marks) Implement the Julia function `euler(f, y0, t0, T, n)` which solves the initial value problem $y'(t) = f(t, y(t))$, $y(t_0) = y_0$ on the interval $[t_0, T]$ using n steps of the explicit Euler method with constant step size, and which returns the vector of length $n + 1$ containing the approximations of y at the grid points $t_0, t_0 + h, \ldots, T$.

A successful test for your implementation would be:

```
julia> @test euler((t, y) -> -y, 1.0, 0.0, 1.0, 2) == [1.0, 0.5, 0.25]
Test Passed
```

Solution:

```
function euler(f, y0, t0, T, n)
  h = (T - t0) / n
  y = zeros(n + 1)
  y[1] = y0
  t = t0
  for k in 1:n
    y[k+1] = y[k] + h * f(t, y[k])
    t += h
  end
  return y
end
```

(b) (3 marks) Using $n = 100$ steps for the test problem above yields `0.3660` for the last component, the exact value is $e^{-1} \approx 0.3679$. What do you expect for the error if you use $n = 200$ steps instead? Justify your answer.

Solution:

The explicit Euler method is a first order method: the global error is proportional to the step size h. Doubling n halves h and therefore the error is expected to be roughly halved, i.e. about 0.0009.

Question 12 (a) Implement a parametric `struct Point{T<:Real}` with two fields `x` and `y` of type `T`, and overload the addition operator `+` for two points (of arbitrary type parameters, adding coordinates).

A successful test for your implementation would be:

```
julia> Point(1, 2) + Point(0.5, 1.5)
Point{Float64}(1.5, 3.5)
```

Solution:

```
struct Point{T<:Real}
  x::T
  y::T
end

Base.:+(a::Point, b::Point) = Point(a.x + b.x, a.y + b.y)
```

(b) With your definition, you evaluate:

```
julia> Point(1, 2.5)
ERROR: MethodError: no method matching Point(::Int64, ::Float64)
The type ‘Point‘ exists, but no method is defined for this combination of argument types

Closest candidates are:
  Point(::T, ::T) where T<:Real
   @ Main REPL[23]:2
```

```
Stacktrace:
 [1] top-level scope
   @ REPL[28]:1
```

Explain in one sentence the reason for this error message.
Solution:
A `MethodError` *occurs: the default constructor requires both coordinates to have the same type* `T`.

(c) Add a method (a constructor) to the code which makes this call return `Point{Float64}(1.0, 2.5)`.
Solution:
Fix using promotion:

```
Point(x::Real, y::Real) = Point(promote(x, y)...)
```

or

```
Point(x::T, y::T2) where {T, T2}=Point(T2(x), y)
```

Question 13 Evaluate the following Julia expressions (each part is worth 1 mark; state `true`/`false` or the resulting type, respectively).

(a) `Int64 <: Integer`

(b) `Vector{Int} <: Vector{Number}`

(c) `Vector{Int} <: AbstractVector{<:Number}`

(d) `isconcretetype(Number)`

(e) `typeof([1, 2.0])`

(f) `supertype(Float64)`

Solution:
(a) `true` *(b)* `false` *(c)* `true`
(d) `false` *(*`Number` *is abstract)*
(e) `Vector{Float64}` *(promotion of the elements)* *(f)* `AbstractFloat`

Question 14 (a) Implement the Julia function `mygcd(a, b)` which computes the greatest common divisor of two non-negative integers using Euclid's algorithm as a recursive function.
Solution:

```
mygcd(a, b) = b == 0 ? a : mygcd(b, a % b)
```

(b) Write down the arguments of all recursive calls when evaluating `mygcd(252, 105)`.
Solution:
$(252, 105) \to (105, 42) \to (42, 21) \to (21, 0)$, *result:* 21.

Question 15 A polynomial $p(x) = c_1 + c_2 x + \cdots + c_{n+1} x^n$ is represented by the vector of its coefficients `p = [c1, c2, ..., c(n+1)]`.

(a) Implement the function `horner(p, x)` which evaluates $p(x)$ using Horner's scheme, i.e. $p(x) = c_1 + x(c_2 + x(c_3 + \dots))$, with a single loop.
A successful test for your implementation would be:

```
julia> @test horner([1, 2, 3], 2) == 17
Test Passed
```

Solution:

```
function horner(p, x)
  r = zero(x)
  for c in reverse(p)
    r = r * x + c
  end
  return r
end
```

(b) How many multiplications does Horner's scheme need for a polynomial of degree n? How many multiplications would a naive evaluation need, in which every power x^k is computed from scratch (by $k-1$ multiplications) and then multiplied by c_{k+1}?
Solution:
Horner: n multiplications. Naive: $\sum_{k=1}^{n} k = n(n+1)/2$ multiplications, i.e. $O(n^2)$.

(c) Use Euclid's algorithm for polynomials to compute a greatest common divisor of $x^2 - 1$ and $x^2 + 2x + 1$. Show the steps.
Solution:
$x^2 + 2x + 1 = 1 \cdot (x^2 - 1) + (2x + 2)$, *so the first remainder is* $2x + 2$.
$x^2 - 1 = \left(\frac{1}{2}x - \frac{1}{2}\right)(2x + 2) + 0$, *so the remainder is* 0.
Therefore the gcd is $2x + 2$, *or normalised (monic)* $x + 1$.

Question 16 (a) Implement the Julia function `estimatepi(n)` which estimates π by a Monte Carlo simulation: draw `n` points uniformly distributed in the unit square $[0, 1]^2$ and count the proportion that fall inside the quarter circle $x^2 + y^2 \leq 1$.

A successful test for your implementation would be:

```
julia> @test abs(estimatepi(10^6) - pi) < 0.01
Test Passed
```

Solution:

```
function estimatepi(n)
  hits = 0
  for _ in 1:n
    hits += rand()^2 + rand()^2 <= 1
  end
  return 4 * hits / n
end
```

(b) How can you make the output of the simulation reproducible?
Solution:
Seed the pseudo-random number generator, e.g. `using Random; Random.seed!(123)`, *before running the simulation.*

Question 17 (a) The exponential distribution with rate $\lambda > 0$ has the cumulative distribution function $F(x) = 1 - e^{-\lambda x}$ for $x \geq 0$. Derive the inverse F^{-1} and explain how to use it, together with a uniform random number, to generate an exponentially distributed random number (inverse transform sampling).
Solution:
Solving $u = 1 - e^{-\lambda x}$ for x gives $F^{-1}(u) = -\ln(1-u)/\lambda$. If U is uniformly distributed on $[0, 1]$ then $X = F^{-1}(U)$ has the cumulative distribution function F. Since $1 - U$ is also uniform, one can equally use $X = -\ln(U)/\lambda$.

(b) Implement the function `exprand(lam)` which returns one exponentially distributed random number with rate `lam` using only `rand()` as the source of randomness. What do you expect for the

output of `mean([exprand(2.0) for _ in 1:10^6])`?
Solution:

```
exprand(lam) = -log(rand()) / lam
```

The mean of the exponential distribution is $1/\lambda$, so the output should be close to 0.5.

Question 18 A single server queue with first-in-first-out service is simulated with a discrete event simulation. Customers arrive at the times 1, 2, and 3, and the service of each customer takes exactly 2.5 time units. The server starts serving a customer as soon as the customer arrives or the previous customer has left, whichever happens later.

(a) List all events (arrival or departure of a customer) in the order in which they are processed by the simulation, together with the event time and the number of customers in the system immediately after the event.
Solution:

time	*event*	*customers in system*
1	*arrival of customer 1*	*1*
2	*arrival of customer 2*	*2*
3	*arrival of customer 3*	*3*
3.5	*departure of customer 1*	*2*
6	*departure of customer 2*	*1*
8.5	*departure of customer 3*	*0*

(b) Which data structure is typically used to store the future events in a simulation engine, and why is it preferable to a simple unsorted vector?
Solution:
A priority queue (e.g. a binary heap) with the event time as the priority. Retrieving the next event (the one with the smallest time) and inserting new events takes $O(\log n)$ operations, whereas for an unsorted vector, finding the next event needs a linear scan, $O(n)$, in every step.

Question 19 Consider the two functions

```
1  function f1(n)
2    s = 0
3    for i in 1:n
4      s += i / 2
5    end
6    return s
7  end
8
9  function f2(n)
10   s = 0.0
11   for i in 1:n
12     s += i / 2
13   end
14   return s
15 end
```

(a) What do `f1(3)` and `f2(3)` return?
Solution:
Both return `3.0` *of type* `Float64`.

(b) One of the functions is type-unstable. Which one, and why?
Solution:
`f1`*: the variable* `s` *starts as an* `Int64` *and becomes a* `Float64` *after the first addition, so its type is not fixed at compile time; the compiler can only infer a* `Union` *of the types (which hampers optimisation). In* `f2` *the variable* `s` *is always* `Float64`*.*

(c) How can you detect this problem using Julia tools, and how do you fix `f1`?
Solution:
With `@code_warntype f1(3)` *(the non-concrete types are highlighted). Fix: initialise* `s = 0.0`*, as in* `f2`*.*

Question 20 Consider the `DataFrame`

```
julia> df = DataFrame(name=["A", "B", "C", "D"],
                      group=["x", "y", "x", "y"],
                      score=[10, 20, 30, 40])
 Row | name    group   score
     | String  String  Int64
-----+----------------------
   1 | A       x          10
   2 | B       y          20
   3 | C       x          30
   4 | D       y          40
```

(with `using DataFrames, Statistics`).

(a) What is the result of `combine(groupby(df, :group), :score => mean => :avg)`?
Solution:
A `DataFrame` *with the columns* `group` *and* `avg` *and two rows:* `("x", 20.0)` *and* `("y", 30.0)`*.*

(b) Which rows are returned by `df[df.score .> 15, :]`?
Solution:
The rows 2, 3, and 4 (names B, C, D), with all three columns.

(c) Add a column `rel` to `df` which contains each score divided by the maximal score.
Solution:
`df.rel = df.score ./ maximum(df.score)`
or `transform(df, :score => (s -> s ./ maximum(s)) => :rel)`*.*
The values are `[0.25, 0.5, 0.75, 1.0]`*.*