SALESFORCE ELIXIR WORKSHOP

IEX

ELIXIR'S

INTERACTIVE

SHELL

IEX - INTERACTIVE ELIXIR SHELL

```
~ » iex
Erlang/OTP 20 [erts-9.0] [source] [64-
bit] [smp:8:8] [ds:8:8:10] [async-
threads:10] [hipe] [kernel-poll:false]
[dtrace]

Interactive Elixir (1.5.0) - press Ctrl
+C to exit (type h() ENTER for help)
iex(1)>
```

HELPFUL FUNCTIONS

- h/1 - prints help for the given module, function or macro
- i/1 - prints information about the data type of any given term
- v/0 - retrieves the last value from the history
- v/1 - retrieves the nth value from the history
- export ERL_AFLAGS="-kernel shell_history enabled" (only OTP 20)

HELPFUL FUNCTIONS

```
iex(2)> h Enum.map

def map(enumerable, fun)

Returns a list where each item is the result of invoking fun on each
corresponding item of enumerable.

For maps, the function expects a key-value tuple.

## Examples

    iex> Enum.map([1, 2, 3], fn(x) -> x * 2 end)
    [2, 4, 6]

    iex> Enum.map([a: 1, b: 2], fn({k, v}) -> {k, -v} end)
    [a: -1, b: -2]
```

HELPFUL FUNCTIONS

```
iex(16)> ["a",
...(16)> #iex:break
** (TokenMissingError) iex:16:
incomplete expression
```

GENERAL

STRINGS

UTF-8 and binaries

```
iex(34)> a = "Hello, Indy” #double quoted
"Hello, Indy"
iex(35)> b = "שלום"
"שלום"
iex(36)> c = "こんにちは世界！"
"こんにちは世界！"
```

STRING INTERPOLATION

```
iex(37)> a <> " " <> b <> " " <> c
"Hello, Indy שלום こんにちは世界！”


iex(37)> “#{a} #{b} #{c}”
"Hello, Indy שלום こんにちは世界！”
```

ATOMS

CONSTANT WHOSE NAME IS ITS VALUE
LIMIT TO NUMBER OF ATOMS

```
iex(58)> :hello
:hello
iex(59)> :"this is an atom!"
:"this is an atom!"
iex(60)> :אחלה
:אחלה
iex(61)> :元気です
:元気です
```

VARIABLES

VARIABLE ASSIGNMENT (ERLANG)

```
1> A = 1.
1
2> A.
1
3> B = 2.
2
4> A = B.
** exception error: no match of right hand side value 2
5> A = 2.
** exception error: no match of right hand side value 2
6>
```

VARIABLE ASSIGNMENT (ELIXIR)

```
Interactive Elixir (1.5.0) - press Ctrl+C to exit (type h()
ENTER for help)
iex(1)> a = 1
1
iex(2)> a
1
iex(3)> b = 2
2
iex(4)> a = 2
2
iex(5)> a = b
2
iex(6)> 1 = a
** (MatchError) no match of right hand side value: 2
```

PINNING VARIABLES

```
iex(67)> a = 1
1
iex(68)> b = a + 1
2
iex(69)> ^a = a
1
iex(70)> ^b = 3
** (MatchError) no match of right hand side value: 3

iex(70)> ^b = 2
2
```

LISTS

Linked List or ordered elements; can include multiple types; faster to prepend than append

```
iex(6)> a = [1,2,3]
[1, 2, 3]
iex(7)> hd a
1
iex(8)> tl a
[2, 3]
iex(17)> Enum.map(a, fn(e) -> IO.puts e end)
1
2
3
[:ok, :ok, :ok]
iex(18)> Enum.map(a, &IO.puts(&1))
1
2
3
[:ok, :ok, :ok]
```

LISTS

```
iex(17)> Enum.map(a, fn(e) -> IO.puts e
end)
1
2
3
[:ok, :ok, :ok]
iex(18)> Enum.map(a, &IO.puts(&1))
1
2
3
[:ok, :ok, :ok]
```

PATTERN MATCHING

```
iex(19)> [a,b] = [1,2]
[1, 2]
iex(20)> a
1
iex(21)> b
2
iex(22)> [a,b,c] = [1,2]
** (MatchError) no match of right hand
side value: [1, 2]
```

CONCAT / SUBTRACT

```
iex(22)> [1,2,3] ++ [4,5,6]
[1, 2, 3, 4, 5, 6]
iex(23)> [1,2,3] -- [1,2]
[3]
iex(32)> [1,2,3] -- [2,3] -- [2,3]
[1, 2, 3]
iex(33)> [1,2,3] -- [1,2,3] -- [2,3] --
[2,3]
[]
```

PATTERN MATCHING

```
iex(19)> list = [1,2]

iex(20)> a = list
iex(21)>
2
iex(22)> [a,b,c] = [1,2]
** (MatchError) no match of right hand
side value: [1, 2]
```

BOOLEANS

true and false; actually :true and :false atoms

```
iex(71)> true and false
false
iex(72)> true and true
true
iex(73)> true or false
true
iex(74)> !true
false
```

NIL

```
iex(51)> nil
nil
iex(52)> is_nil(nil)
true
iex(53)> 1 || nil
1
iex(54)> nil || 1
1
```

MAPS

Collection of Key-value pairs

```
iex(49)> map = %{"hello": "world"}
%{hello: "world"}
iex(50)> map.hello
"world"
iex(51)> map[:hello]
"world"
iex(52)> map = %{map | "hello": “Indy"} # Update Value
%{hello: "Indy"}
iex(53)> map.hello
"Indy"
```

TUPLES

ORDERED LIST OF ELEMENTS OF FIXED SIZE

```
iex(72)> a = {1,2}
{1, 2}
iex(73)> elem(a,0)
1
iex(74)> Tuple.append(a, 3)
{1, 2, 3}
iex(75)> {:ok, "value"}
{:ok, "value"}
iex(76)> {:error, "error"}
{:error, "error"}
```

RANGES

```
iex(55)> 1..10
1..10
iex(56)> Enum.to_list(1..10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
iex(57)> Enum.to_list(1..-5)
[1, 0, -1, -2, -3, -4, -5]
```

PROBLEM

Create a range from 1 to 100. Then use Enum.filter/2 to produce a list of all even integers.

PROBLEM

Create a range from 1 to 100. Then use Enum.filter/2 to produce a list of all even integers.

```
iex(84)> range = 1..100
1..100
iex(85)> Enum.filter(range, fn x -> rem(x,2) == 0 end)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
30, 32, 34, 36, 38, 40, 42,
 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
70, 72, 74, 76, 78, 80, 82,
 84, 86, 88, 90, 92, 94, 96, 98, 100]
iex(86)>
```

STREAM

Lazy evaluation of a collection. Evaluated with an Enum function call.

```
iex(70)> stream = Stream.map([1,2,3], fn(x)
-> x * 2 end)

#Stream<[enum: [1, 2, 3], funs:
[#Function<46.40091930/1 in Stream.map/2>]]>

iex(71)> Enum.reverse(stream)
[6, 4, 2]
```

FOR

Comprehensions allow you to build data structures from enumerables or bitstrings

```
iex(76)> for x <- [1,2,3,4], do: x + 1
[2, 3, 4, 5]

iex(77)> for x <- [1,2], y <- [3,4], do: x
+ y
[4, 5, 5, 6]
```

FOR

Comprehensions allow you to build data structures from enumerables or bitstrings

```
iex(76)> for x <- [1,2,3,4], do: x + 1
[2, 3, 4, 5]

iex(77)> for x <- [1,2], y <- [3,4], do: x
+ y
[4, 5, 5, 6]
```

KEYWORD LISTS

MAPSETS

STRUCTS

MATH OPERATIONS

PATTERN MATCHING

COND

```
iex(12)> a = 6


iex(13)> cond do
...(13)> a + 1 == 8 -> 0
...(13)> a + 1 == 7 -> 1
...(13)> true       -> :error
...(13)> end
```

CASE

```
iex(33)> status = :finished
:finished
iex(34)> case status do
...(34)> :not_started -> 0
...(34)> :in_progress -> 1
...(34)> :finished    -> 2
...(34)> _            -> :invalid_state
...(34)> end
2
```

ANONYMOUS FUNCTIONS

ANONYMOUS FUNCTIONS

```
iex(37)> func = fn -> :anonymous end
#Function<20.99386804/0 in :erl_eval.expr/5>

iex(38)> func.()
:anonymous

iex(39)> subtraction = fn(a,b) -> a - b end
#Function<12.99386804/2 in :erl_eval.expr/5>

iex(40)> subtraction.(1,2)
-1
```

PROBLEM

Write an anonymous function that calculates the amount of sales tax for an item assuming sales tax is 8.25%.

PATTERN MATCHING

```
iex(41)> ticket_status = fn
...(41)> :not_started -> 0
...(41)> :in_progress -> 1
...(41)> :finished    -> 2
...(41)>  _           -> :invalid_state
...(41)> end
#Function<6.99386804/1 in :erl_eval.expr/5>

iex(42)> ticket_status.(:finished)
2
```

PROBLEM

Alice and Bob pay different tax rates. Alice pays 5%, Bob pays 7%. Write an anonymous function that takes two arguments - an atom (:alice or :bob) and an integer - that calculates the total amount for integer price.

MODULES

Collection of named functions

```
iex(43)> defmodule MyModule do
...(43)> @moduledoc """
...(43)> This contains my functions
...(43)> """
...(43)> def hello do
...(43)> :hello
...(43)> end
...(43)> end

{:module, MyModule,
 <<70, 79, 82, 49, 0, 0, 3, 160, 66, 69, 65, 77, 65, 116, 85, 56, 0, 0, 0, 87,
   0, 0, 0, 8, 15, 69, 108, 105, 120, 105, 114, 46, 77, 121, 77, 111, 100, 117,
   108, 101, 8, 95, 95, 105, 110, 102, 111, ...>>, {:hello, 0}}

iex(44)> MyModule.hello
:hello
```

FUNCTIONS

```
defmodule MyModule do
   @doc """
     Sums two integers
   """
   def sum_two_integers(a,b) do
      a + b
   end
end
```

GUARD CLAUSES

```
iex(36)> defmodule MyGuard do
...(36)> def my_add(a,b) when
is_integer(a) and is_integer(b) do
...(36)> a + b
...(36)> end
...(36)> end
```

BETTER ERRORS

```
iex(36)> MyGuard.my_add(1,"b")
** (FunctionClauseError) no function clause matching in
MyGuard.my_add/2

    The following arguments were given to A.my_add/2:

        # 1
        1

        # 2
        "b"

    iex:35: MyGuard.my_add/2
```

LET IT CRASH

GUARD CLAUSES

```
iex(36)> defmodule MyGuard do
...(36)> def my_add(a,b) when is_integer(a) and
is_integer(b) do
...(36)> a + b
...(36)> end
...(36)> def my_add(a,b) do
...(36)> {:error, :invalid_inputs}
...(36)> end
...(36)> end
iex(37)> MyGuard.my_add("b", [])
{:error, :invalid_inputs}
```

MULTIPLE FUNCTION DEFINITIONS

```
iex(12)> defmodule MultipleFunctions do
...(12)> def return_status(input) when input < 5 do
...(12)> :incomplete
...(12)> end
...(12)> def return_status(input) when input == 0 do
...(12)> :not_started
...(12)> end
...(12)> def return_status(_input), do: :finished
...(12)> end
iex(13)> MultipleFunctions.return_status(0)
:incomplete
iex(14)> MultipleFunctions.return_status(:nl)
:finished
iex(15)> MultipleFunctions.return_status(nil)
:finished
```

PROBLEM

Open functions/multiple_function_defs.exs and create multiple functions that handle the cases listed in the @moduledoc

PIPE OPERATOR

```
initial = 1
add_two = initial + 2
multiply_by_six = add_two * 6


1
|> add_two
|> multiply_by_six
```

OBSERVER

```
Interactive Elixir (1.5.0) - press Ctrl
+C to exit (type h() ENTER for help)
iex(1)> :observer.start()
:ok
iex(2)>
```

nonode@nohost

System | Load Charts | Memory Allocators | Applications | Processes | Ports | Table Viewer | Trace Overview

System and Architecture

System Version:	20
ERTS Version:	9.0
Compiled for:	x86_64-apple-darwin15.6.0
Emulator Wordsize:	8
Process Wordsize:	8
SMP Support:	true
Thread Support:	true
Async thread pool size:	10

Memory Usage

Total:	27 MB
Processes:	6399 kB
Atoms:	379 kB
Binaries:	329 kB
Code:	8939 kB
ETS:	841 kB

CPU's and Threads

Logical CPU's:	8
Online Logical CPU's:	8
Available Logical CPU's:	unknown
Schedulers:	8
Online schedulers:	8
Available schedulers:	8

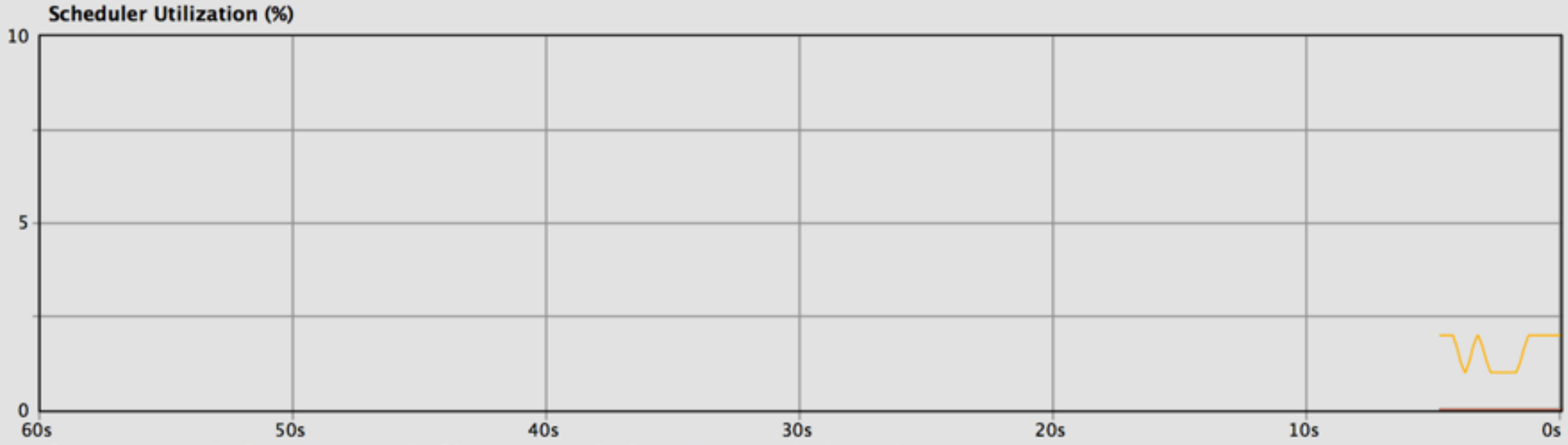
Statistics

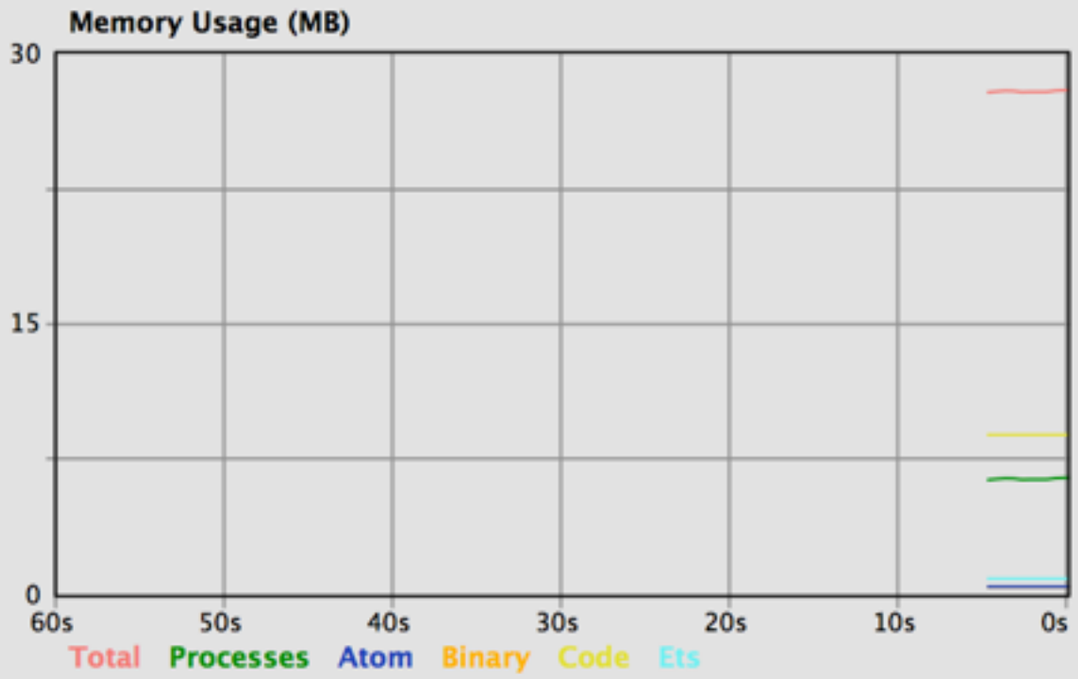
Up time:	12 Mins
Max Processes:	262144
Processes:	65
Run Queue:	0
IO Input:	7559 kB
IO Output:	137 kB

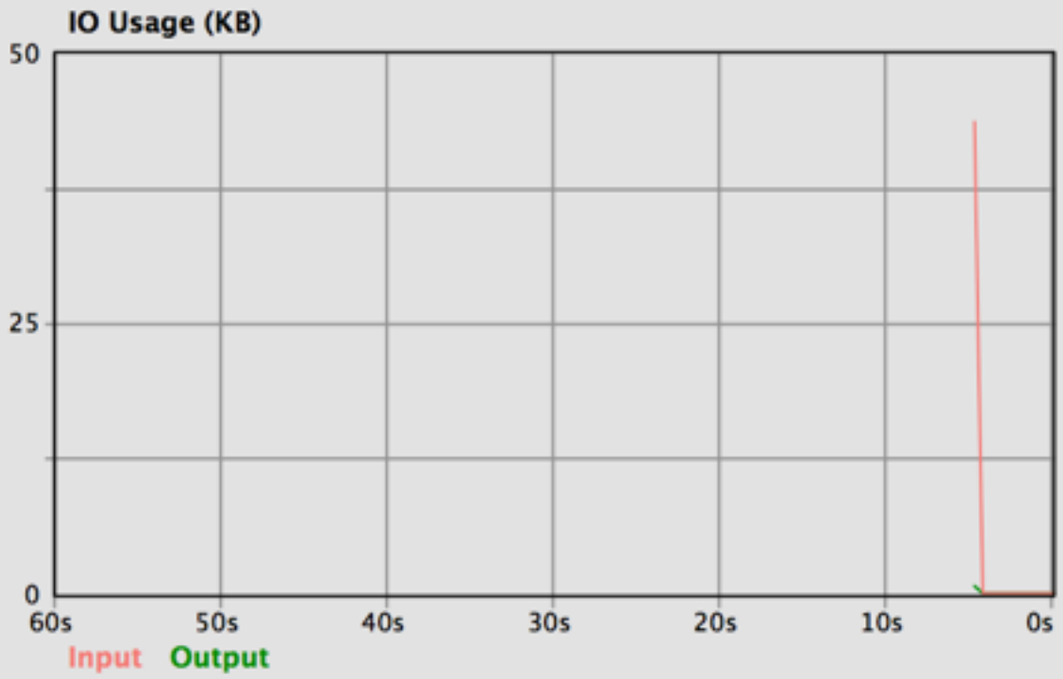
Number of Processes: 61

nonode@nohost

System | Load Charts | Memory Allocators | Applications | Processes | Ports | Table Viewer | Trace Overview



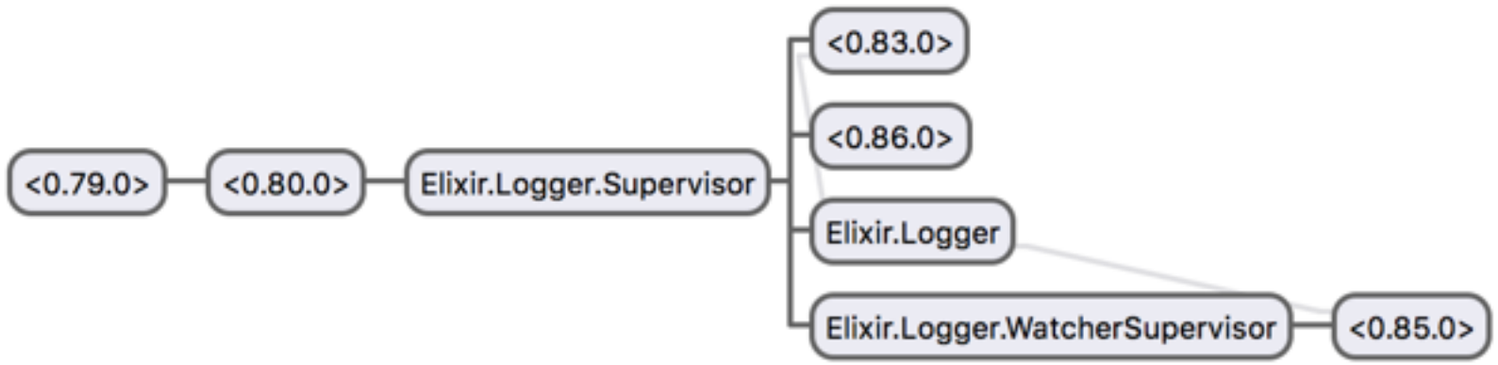




nonode@nohost

System | Load Charts | Memory Allocators | Applications | Processes | Ports | Table Viewer | Trace Overview

- elixir
- iex
- kernel
- logger



nonode@nohost

System | Load Charts | Memory Allocators | Applications | Processes | Ports | Table Viewer | Trace Overview

Pid	Name or Initial Func	Reds	Memory	MsgQ	Current Function
<0.6.0>	erl_prim_loader	583209	197008	0	prim_file:drv_get_response/1
<0.38.0>	code_server	151126	284744	0	erl_prim_loader:request/1
<0.33.0>	application_controller	52332	372592	0	gen_server:loop/7
<0.95.0>	observer_sys_wx:init/1	24152	142944	0	wx_object:loop/6
<0.61.0>	disk_log:init/2	14203	88800	0	disk_log:loop/1
<0.90.0>	observer	13650	143136	0	proc_lib:sync_wait/2
<0.53.0>	group:server/3	10882	973352	0	group:server_loop/3
<0.98.0>	observer_alloc_wx:init/1	4504	10824	0	wx_object:loop/6
<0.46.0>	file_server_2	4347	24800	0	gen_server:loop/7
<0.97.0>	observer_perf_wx:init/1	3595	11968	0	wx_object:loop/6
<0.51.0>	user_drv	2558	13848	0	user_drv:server_loop/6
<0.37.0>	kernel_sup	2093	6312	0	gen_server:loop/7
<0.99.0>	observer_app_wx:init/1	2044	8952	0	wx_object:loop/6
<0.88.0>	Elixir.IEx.Evaluator:init/4	2029	12000	0	proc_lib:sync_wait_mon/2
<0.0.0>	init	1830	26552	0	init:loop/1
<0.91.0>	wxe_server:init/1	1179	12584	0	gen_server:loop/7
<0.92.0>	wxe_master	831	10864	0	gen_server:loop/7
<0.32.0>	error_logger	829	20760	0	gen_event:fetch_msg/6
<0.100.0>	observer_pro_wx:init/1	664	6064	1	wxe_util:rec/1
<0.81.0>	Elixir.Logger.Supervisor	502	9040	0	gen_server:loop/7
<0.82.0>	Elixir.Logger	486	7088	0	gen_event:fetch_msg/6
<0.55.0>	kernel_safe_sup	378	7128	0	gen_server:loop/7
<0.74.0>	Elixir.IEx.Supervisor	347	7088	0	gen_server:loop/7
<0.63.0>	erlang:apply/2	297	5936	0	Elixir.IEx.Server:wait_eval/3
<0.67.0>	application_master:start_it/4	297	7032	0	application_master:loop_it/4
<0.44.0>	inet_db	272	2928	0	gen_server:loop/7

PROCESSES

```
iex(39)> current_process = self()
#PID<0.92.0>
iex(40)> Process.alive? current_process
true
iex(42)> Process.monitor current_process
#Reference<0.3226719017.4050386945.219048>
iex(43)> Process.info current_process
[current_function: {Process, :info, 1}, initial_call:
{:proc_lib, :init_p, 5},
 status: :running, message_queue_len: 0, messages: [],
links: [],
 dictionary: ["$ancestors": [#PID<0.63.0>],
  "$initial_call": {IEx.Evaluator, :init, 4},
  iex_history: %IEx.History.State{queue: {[{42,
      'Process.monitor current_process\n',
     #Reference<0.3226719017.4050386945.219048>}
```

LOADING FILES

```
» iex file_name.exs

iex(2)> c "recursion_sum.exs"
warning: redefining module Recursion
(current version defined in memory)
  recursion_sum.exs:1

[Recursion]
iex(3)>
```

RECURSION

```
iex(9)> defmodule Recursion do
...(9)> def list_to_string(list, acc \\
"")
...(9)> def list_to_string([], acc), do:
acc
...(9)> def list_to_string([h|t], acc) do
...(9)> list_to_string(t, acc <> h)
...(9)> end
...(9)> end
```

RECURSION

```
iex(10)> Recursion.list_to_string([])
""
iex(11)>
Recursion.list_to_string(["a","b","c"])
"abc"
iex(12)> Recursion.list_to_string([1])

** (ArgumentError) argument error
    :erlang.bit_size(1)
    iex:13: Recursion.list_to_string/2
```

PROBLEM A

Implement recursions/problem_a.exs which adds enumerates over a list and sums the list

PROBLEM C

Implement recursions/problem_c.exs using tail call recursion

TAIL CALL OPTIMIZATION

Instead of adding a new function to the stack, the same function is returned with new values. It's much more efficient as you avoid adding stack frames.

non-tail call: `fib(n-1) + fib(n-2)`

tail-call: `fib(n-1, next + acc, next)`

PROBLEM D

Implement recursions/problem_d.exs using tail call recursion

MIX

```
» mix new my_first_app
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/my_first_app.ex
* creating test
* creating test/test_helper.exs
* creating test/my_first_app_test.exs

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd my_first_app
    mix test

Run "mix help" for more commands.
```

```
» mix help
mix                         # Runs the default task (current:
"mix run")
mix compile                 # Compiles source files
mix deps                    # Lists dependencies and their
status
mix deps.clean              # Deletes the given dependencies'
files
mix deps.get                # Gets all out of date
dependencies
mix hex.outdated            # Shows outdated Hex deps for the
current project
mix new                     # Creates a new Elixir project
mix phoenix.new             # Creates a new Phoenix v1.2.4
application
mix run                     # Runs the given file or
expression
mix test                    # Runs a project's tests
iex -S mix                  # Starts IEx and runs the default
task
```

```
» mix new my_first_sup_app --sup
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/my_first_sup_app.ex

* creating lib/my_first_sup_app/application.ex #supervisison tree

* creating test
* creating test/test_helper.exs
* creating test/my_first_sup_app_test.exs

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd my_first_sup_app
    mix test

Run "mix help" for more commands.
```

```
defmodule MyFirstSupApp.Application do
  # See https://hexdocs.pm/elixir/Application.html
  # for more information on OTP Applications
  @moduledoc false

  use Application

  def start(_type, _args) do
    # List all child processes to be supervised
    children = [
      # Starts a worker by calling:
MyFirstSupApp.Worker.start_link(arg)
      # {MyFirstSupApp.Worker, arg},
    ]

    # See https://hexdocs.pm/elixir/Supervisor.html
    # for other strategies and supported options
    opts = [strategy: :one_for_one, name:
MyFirstSupApp.Supervisor]
    Supervisor.start_link(children, opts)
  end
end
```

SUPERVISION STRATEGIES

https://hexdocs.pm/elixir/Supervisor.html

- **:one_for_one -** if a child process terminates, only that process is restarted.
- **:one_for_all -** if a child process terminates, all other child processes are terminated and then all child processes (including the terminated one) are restarted.
- **:rest_for_one -** if a child process terminates, the "rest" of the child processes, i.e., the child processes after the terminated one in start order, are terminated. Then the terminated child process and the rest of the child processes are restarted.
- **:simple_one_for_one -** similar to :one_for_one but suits better when dynamically attaching children. This strategy requires the supervisor specification to contain only one child. Many functions in this module behave slightly differently when this strategy is used.

RESTART STRATEGIES

https://hexdocs.pm/elixir/Supervisor.Spec.html#t:restart/0

- :permanent - the child process is always restarted
- :temporary - the child process is never restarted (not even when the supervisor's strategy is :rest_for_one or :one_for_all)
- :transient - the child process is restarted only if it terminates abnormally, i.e., with an exit reason other than :normal, :shutdown or {:shutdown, term}

PROBLEM

in mix/my_first_app, open test/my_first_app_test.exs and fill out the tests and make sure they pass

ONLINE RESOURCES

ELIXIR LANG

ELIXIR SCHOOL

ELIXIR FORUM

ELIXIR SLACK & IRC

ELIXIR RADAR

ELIXIRWEEKLY

BOOKS

ELIXIR IN ACTION

PROGRAMMING ELIXIR

METAPROGAMMING ELIXIR

PROGRAMMING PHOENIX

LITTLE ELIXIR & OTP GUIDEBOOK

ADOPTING ELIXIR

QUESTIONS?

@bgmarx