PureScript is a small strongly, statically typed language compiling to Javascript.

- Learn more at http://www.purescript.org/
- Documentation: http://pursuit.purescript.org/
- Book: Purescript by Example, https://leanpub.com/purescript/

All the noncommented lines of code can be run in the PSCI REPL, though some will require the `--multi-line-mode` flag.

```
--
-- 1. Primitive datatypes that corresponds to their Javascript
-- equivalents at runtime.

import Prelude
-- Numbers
1.0 + 7.2*5.5 :: Number -- 40.6
-- Ints
1 + 2*5 :: Int -- 11
-- Types are inferred, so the following works fine
9.0/2.5 + 4.4 -- 8.0
-- But Ints and Numbers don't mix, so the following won't
5/2 + 2.5 -- Expression 2.5 does not have type Int
-- Hexadecimal literals
0xff + 1 -- 256
-- Unary negation
6 * -3 -- -18
6 * negate 3 -- -18
-- Modulus, from purescript-math (Math)
3.0 % 2.0 -- 1.0
4.0 % 2.0 -- 0.0
-- Inspect the type of an expression in psci
:t 9.5/2.5 + 4.4 -- Prim.Number

-- Booleans
true :: Boolean -- true
false :: Boolean -- false
-- Negation
not true -- false
23 == 23 -- true
1 /= 4 -- true
1 >= 4 -- false
-- Comparisions < <= > >=
-- are defined in terms of compare
compare 1 2 -- LT
compare 2 2 -- EQ
compare 3 2 -- GT
-- Conjunction and Disjunction
true && (9 >= 19 || 1 < 2) -- true

-- Strings
"Hellow" :: String -- "Hellow"
-- Multiline string without newlines, to run in psci use the --multi-line-mode flag
```

```
"Hellow\
\orld" -- "Helloworld"
-- Multiline string with newlines
"""Hello
world""" -- "Hello\nworld"
-- Concatenate
"such " ++ "amaze" -- "such amaze"

--
-- 2. Arrays are Javascript arrays, but must be homogeneous

[1,1,2,3,5,8] :: Array Number -- [1,1,2,3,5,8]
[true, true, false] :: Array Boolean -- [true,true,false]
-- [1,2, true, "false"] won't work
-- `Cannot unify Prim.Int with Prim.Boolean`
-- Cons (prepend)
1 : [2,4,3] -- [1,2,4,3]

-- Requires purescript-arrays (Data.Array)
-- and purescript-maybe (Data.Maybe)

-- Safe access return Maybe a
head [1,2,3] -- Just (1)
tail [3,2,1] -- Just ([2,1])
init [1,2,3] -- Just ([1,2])
last [3,2,1] -- Just (1)
-- Random access - indexing
[3,4,5,6,7] !! 2 -- Just (5)
-- Range
1..5 -- [1,2,3,4,5]
length [2,2,2] -- 3
drop 3 [5,4,3,2,1] -- [2,1]
take 3 [5,4,3,2,1] -- [5,4,3]
append [1,2,3] [4,5,6] -- [1,2,3,4,5,6]

--
-- 3. Records are Javascript objects, with zero or more fields, which
-- can have different types.
-- In psci you have to write `let` in front of the function to get a
-- top level binding.
let book = {title: "Foucault's pendulum", author: "Umberto Eco"}
-- Access properties
book.title -- "Foucault's pendulum"

let getTitle b = b.title
-- Works on all records with a title (but doesn't require any other field)
getTitle book -- "Foucault's pendulum"
getTitle {title: "Weekend in Monaco", artist: "The Rippingtons"} -- "Weekend in Monaco"
-- Can use underscores as shorthand
_.title book -- "Foucault's pendulum"
-- Update a record
let changeTitle b t = b {title = t}
getTitle (changeTitle book "Ill nome della rosa") -- "Ill nome della rosa"
```

```
--
-- 4. Functions
-- In psci's multiline mode
let sumOfSquares :: Int -> Int -> Int
    sumOfSquares x y = x*x + y*y
sumOfSquares 3 4 -- 25
let myMod x y = x % y
myMod 3.0 2.0 -- 1.0
-- Infix application of function
3 `mod` 2 -- 1

-- function application has higher precedence than all other
-- operators
sumOfSquares 3 4 * sumOfSquares 4 5 -- 1025

-- Conditional
let abs' n = if n>=0 then n else -n
abs' (-3) -- 3

-- Guarded equations
let abs'' n | n >= 0    = n
            | otherwise = -n

-- Pattern matching

-- Note the type signature, input is a list of numbers. The pattern matching
-- destructures and binds the list into parts.
-- Requires purescript-lists (Data.List)
let first :: forall a. List a -> a
    first (Cons x _) = x
first (toList [3,4,5]) -- 3
let second :: forall a. List a -> a
    second (Cons _ (Cons y _)) = y
second (toList [3,4,5]) -- 4
let sumTwo :: List Int -> List Int
    sumTwo (Cons x (Cons y rest)) = x + y : rest
fromList (sumTwo (toList [2,3,4,5,6])) :: Array Int -- [5,4,5,6]

-- sumTwo doesn't handle when the list is empty or there's only one element in
-- which case you get an error.
sumTwo [1] -- Failed pattern match

-- Complementing patterns to match
-- Good ol' Fibonacci
let fib 1 = 1
    fib 2 = 2
    fib x = fib (x-1) + fib (x-2)
fib 10 -- 89

-- Use underscore to match any, where you don't care about the binding name
let isZero 0 = true
    isZero _ = false

-- Pattern matching on records
```

```
let ecoTitle {author = "Umberto Eco", title = t} = Just t
    ecoTitle _ = Nothing

ecoTitle book -- Just ("Foucault's pendulum")
ecoTitle {title: "The Quantum Thief", author: "Hannu Rajaniemi"} -- Nothing
-- ecoTitle requires both field to type check:
ecoTitle {title: "The Quantum Thief"} -- Object lacks required property "author"

-- Lambda expressions
(\x -> x*x) 3 -- 9
(\x y -> x*x + y*y) 4 5 -- 41
let sqr = \x -> x*x

-- Currying
let myAdd x y = x + y -- is equivalent with
let myAdd' = \x -> \y -> x + y
let add3 = myAdd 3
:t add3 -- Prim.Int -> Prim.Int

-- Forward and backward function composition
-- drop 3 followed by taking 5
(drop 3 >>> take 5) (1..20) -- [4,5,6,7,8]
-- take 5 followed by dropping 3
(drop 3 <<< take 5) (1..20) -- [4,5]

-- Operations using higher order functions
let even x = x `mod` 2 == 0
filter even (1..10) -- [2,4,6,8,10]
map (\x -> x + 11) (1..5) -- [12,13,14,15,16]

-- Requires purescript-foldable-traversabe (Data.Foldable)

foldr (+) 0 (1..10) -- 55
sum (1..10) -- 55
product (1..10) -- 3628800

-- Testing with predicate
any even [1,2,3] -- true
all even [1,2,3] -- false
```