# JavaScript Type Inference

CLASS : ADVANCED PROGRAMMING LANGUAGE PRINCIPLES
NAME : YOGESH DIXIT
ID : 010713963

# Type Inference

- Type Inference in a programming language means making a logical conclusion about the type of an expression
- In statically typed languages like JAVA, C and Haskell type of an expression is known at compile time
- Statically typed languages such as C and JAVA lack type inference, thus it is required that programmers declare the types when they write programs
- Statically typed languages like Haskell uses type inferencing and it is not required for programmer to declare types while writing

# Type Inference

- In dynamically typed languages like JavaScript and Ruby type is associated with each value rather than expression and is known only at runtime
- What does this actually mean?
- In JavaScript as we all know, a variable at one time bound to a number may later be rebound to some different type of value such as string.

# Type Inferencing Example (Haskell)

- In the given example, let's try to deduce how type inferencing is performed in Haskell
- mymap is a function with two arguments (therefore of the form a -> b-> c)
- Patterns [ ] and (first:rest) always match lists, so second argument is list [d] (list with members of type d)

```
mymap f [] = []
mymap f (first:rest) = f first : map f rest

addone :: Int -> Int
addone x = x + 1

main :: IO ()
main = do
  putStrLn $ show $ mymap addone [1,2,3,4]
```

# Type Inferencing Example (Haskell)

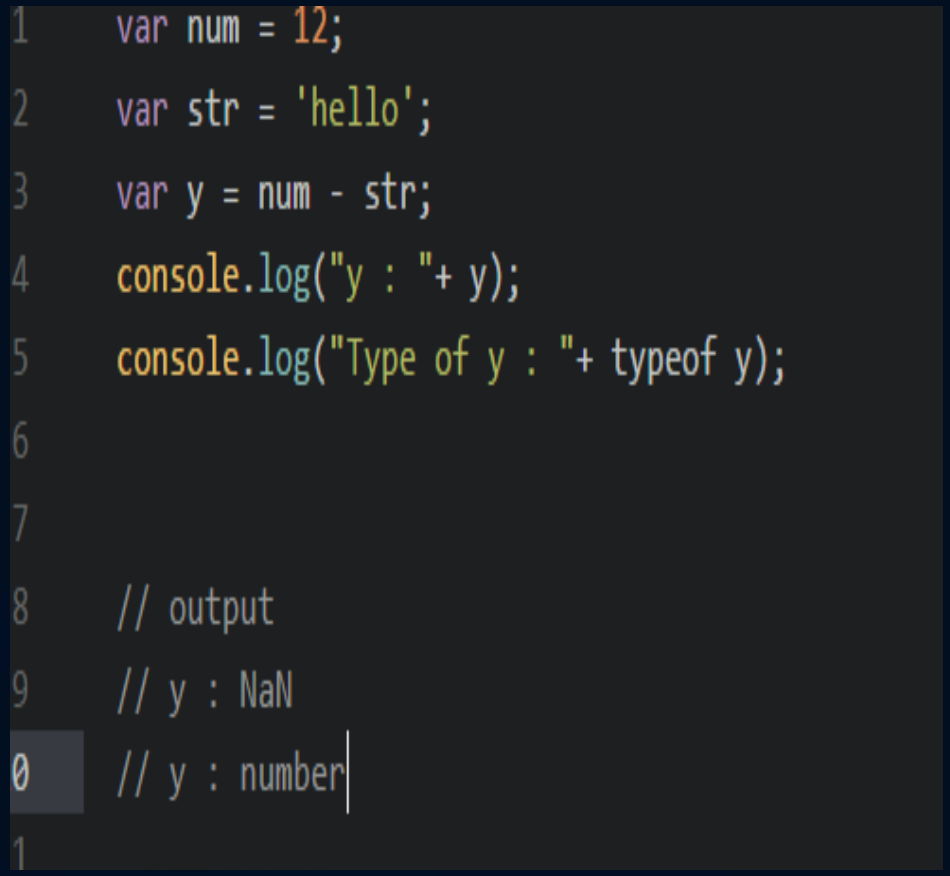
- First argument f is applied to the "first" which must be of type d.
- Thus f is a function of type d -> e
- Finally mymap produces a list of whatever function f produces, thus [e]
- Putting all the parts together we obtain signature of mymap would be mymap :: (a -> b) -> [d] -> [b]

```haskell
mymap f [] = []
mymap f (first:rest) = f first : map f rest

addone :: Int -> Int
addone x = x + 1

main :: IO ()
main = do
  putStrLn $ show $ mymap addone [1,2,3,4]
```

- Hindley-Milner type inference algorithm is well known algorithm for type inferencing in Haskell
- Origin of this algorithm is from the works of Haskell Curry and Robert Feys for simply typed lambda calculus
- Roger Hindley extended this work and Robert Milner provided an equivalent algorithm
- In 1982, Luis Damas finally proved that Milner's algorithm is complete and extended it to support systems with polymorphic references.
- Thus Hindley-Milner algorithm is also known as Damas-Milner or Damas-Hindley-Milner algorithm.

# JavaScript without Type Inference

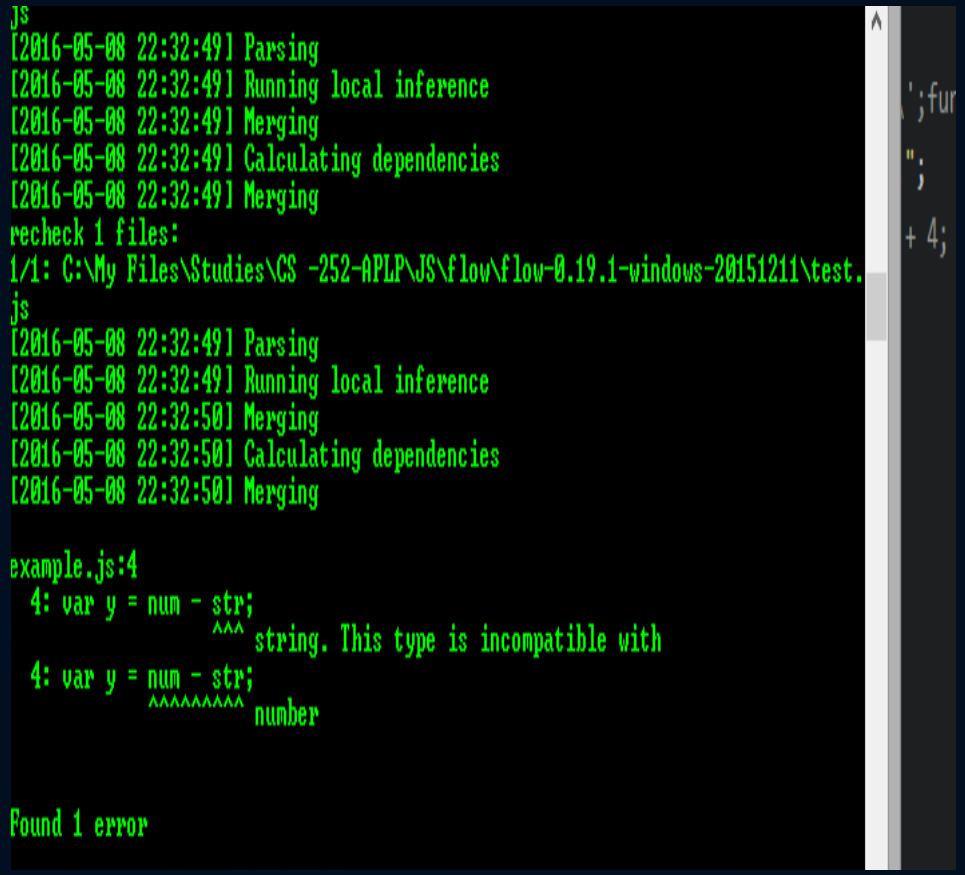
- As there are no type annotations and no type inferencing in JavaScript
- In the program as shown here we get some weird output which can lead runtime errors in more complex programs or situations
- We can avoid such type errors by simply performing type inference on JS code



```
var num = 12;
var str = 'hello';
var y = num - str;
console.log("y : "+ y);
console.log("Type of y : "+ typeof y);



// output
// y : NaN
// y : number
```

# Type Inference in JavaScript using Flow JS

- Flow is a static type checker designed to quickly find type errors in JavaScript applications
- Flow relies heavily on Type Inference to find type errors
- It precisely tracks the types of the variables as they flow through the program



# Type Inferencing in JavaScript (Project)

- Perform Type Inferencing on a given JavaScript code using AST
- Construct AST using Reflect JS/ Esprima library
- Once the AST is constructed walk through the AST to infer the types of the expressions using the types of the values involved in the expression
- Use the range and location objects associated with every node of the AST to precisely determine the type errors and the location at which the type error is present

# Type Inference in JavaScript (my utility program output example)

```
1    var num = 12;
2    var str = 'hello';
3    var y = num - str;
4    console.log("y : "+ y);
5    console.log("Type of y : "+ typeof y);
6
7
8    // output
9    // y : NaN
0    // y : number
1
```

```
C:\My Files\Studies\CS -252-APLP\JS\TypeInferencingProject>clear


C:\My Files\Studies\CS -252-APLP\JS\TypeInferencingProject>node TypeInference.js

Inferred type for num is Numeric
Inferred type for str is String
Type Error....
[ { rangeObj: [ 4, 12 ], inferredType: 'Numeric', name: 'num' },
  { rangeObj: [ 17, 30 ], inferredType: 'String', name: 'str' } ]

C:\My Files\Studies\CS -252-APLP\JS\TypeInferencingProject>
```

# Benefits

- Reduce or eliminate possible runtime type errors from the program.
- Without using type annotations achieve the better results
- More reliable code

# Thank You