Haskell Crash Course Part I

From the Lambda Calculus to Haskell

+ builtin Int, Char, ...
+ types
+ run-time
+ compiler
+ ...

Programming in Haskell

Computation by Calculation
Substituting equals by equals

Computation via **Substituting Equals by Equals**

\[(1 + 3) \times (4 + 5)\]

\[=\Rightarrow 4 \times (4 + 5)\]

\[\text{-- subst } 1 + 3 = 4\]

\[=\Rightarrow 4 \times 9\]

\[\text{-- subst } 4 + 5 = 9\]

\[=\Rightarrow 36\]

\[\text{-- subst } 4 \times 9 = 36\]

\[f_{x}y = x + y\]
Computation via Substituting Equals by Equals

Equality-Substitution enables Abstraction via Pattern Recognition

Abstraction via Pattern Recognition
Repeated Expressions

\[ x \times (y + z) \]

\[
\begin{align*}
31 & \times (42 + 56) \\
70 & \times (12 + 95) \\
90 & \times (68 + 12)
\end{align*}
\]

Recognize Pattern as \( \lambda \)-function

\[ \text{pat} = \lambda x \ y \ z \to x \times (y + z) \]

Equivalent Haskell Definition

\[ \text{pat} \ x \ y \ z = x \times (y + z) \]

Function Call is Pattern Instance

\[
\begin{align*}
\text{pat} \ 31 \ 42 \ 56 & \Rightarrow 31 \times (42 + 56) \Rightarrow 31 \times 98 \Rightarrow 3038 \\
\text{pat} \ 70 \ 12 \ 95 & \Rightarrow 70 \times (12 + 95) \Rightarrow 70 \times 107 \Rightarrow 7490 \\
\text{pat} \ 90 \ 68 \ 12 & \Rightarrow 90 \times (68 + 12) \Rightarrow 90 \times 80 \Rightarrow 7200
\end{align*}
\]

**Key Idea:** Computation is *substitute* equals by equals.
Programming in Haskell

Substitute Equals by Equals

That's it! (Do not think of registers, stacks, frames etc.)

Elements of Haskell
- Core program element is an **expression**
- Every *valid* expression has a **type** (determined at compile-time)
- Every *valid* expression reduces to a **value** (computed at run-time)

**Ill-typed** expressions are rejected at **compile-time** before execution
  - *like* in Java
  - *not like* \(\lambda\)-calculus or Python ...

---

**The Haskell Eco-System**
Batch compiler: \texttt{ghc} Compile and run large programs

Interactive Shell \texttt{ghci} Shell to interactively run small programs online (https://repl.it/languages/haskell)

Build Tool \texttt{stack} Build tool to manage libraries etc.

\textit{Interactive Shell: \texttt{ghci}}

\$ \texttt{stack ghci}

:load file.hs
:type expression
:info variable
A Haskell Source File

A sequence of top-level definitions \( x_1, x_2, \ldots \)

- Each has type \( \text{type}_1, \text{type}_2, \ldots \)
- Each defined by expression \( \text{expr}_1, \text{expr}_2, \ldots \)

\[
\begin{align*}
x_1 &:: \text{type}_1 \\
x_1 &\equiv \text{expr}_1 \\
x_2 &:: \text{type}_2 \\
x_2 &\equiv \text{expr}_2 \\
\end{align*}
\]
Basic Types

ex1 :: Int
    ex1 = 31 * (42 + 56) -- this is a comment

ex2 :: Double
    ex2 = 3 * (4.2 + 5.6) -- arithmetic operators "overloaded"

ex3 :: Char
    ex3 = 'a', 'b' -- 'a', 'b', 'c', etc. built-in `Char` values

ex4 :: Bool
    ex4 = True -- True, False are builtin Bool values

ex5 :: Bool
    ex5 = False
**QUIZ: Basic Operations**

```haskell
ex6 :: Int
ex6 = 4 + 5

ex7 :: Int
ex7 = 4 * 5

ex8 :: Bool
ex8 = 5 > 4

quiz :: ?
quiz = if ex8 then ex6 else ex7
```

What is the type of quiz?

A. Int
B. Bool
C. Error!
**QUIZ: Basic Operations**

ex6 :: Int
ex6 = 4 + 5

ex7 :: Int
ex7 = 4 * 5

ex8 :: Bool
ex8 = 5 > 4  *TRUE*

quiz :: ???
quiz = if ex8 then ex6 else ex7

What is the value of quiz?

A. 9
Function Types

In Haskell, a function is a value that has a type

$$ A \rightarrow B $$

A function that

- takes input of type $A$
- returns output of type $B$

For example

```haskell
isPos :: Int -> Bool
isPos = \n  \x -> (x > 0)
```
Define **function-expressions** using \ like in \textit{\lambda}-calculus!

But Haskell also allows us to put the parameter on the \textit{left}

\[
\text{isPos :: Int }\rightarrow\text{ Bool}
\]

\[
isPos\ n = (x > 0)
\]

(Meaning is \textbf{identical} to above definition with \textbackslash\textit{n} \rightarrow \ldots)

---

**Multiple Argument Functions**

A function that

- takes three \textit{inputs} A1, A2 and A3
- returns one \textit{output} B has the type

\[
A1 \rightarrow A2 \rightarrow A3 \rightarrow B
\]

For example
\[ \text{pat} :: \text{Int} \to \text{Int} \to \text{Int} \to \text{Int} \]
\[ \text{pat} = \{x \ y \ z \to x \ast (y + z)\} \]

which we can write with the params on the left as

\[ \text{pat} :: \text{Int} \to \text{Int} \to \text{Int} \to \text{Int} \]
\[ \text{pat} \ x \ y \ z = x \ast (y + z) \]

**QUIZ**

What is the type of quiz?

\[ \text{quiz} :: \text{???} \]
\[ \text{quiz} \ x \ y = (x + y) > 0 \]

A. Int \to Int

B. Int \to \text{Bool}
C. Int -> Int -> Int
D. Int -> Int -> Bool
E. (Int, Int) -> Bool

**Function Calls**

A function call is exactly like in the $\lambda$-calculus

```
e1 e2
```

where e1 is a function and e2 is the argument. For example

```
>>> isPos 12
True

>>> isPos (0 - 5)
False
```
Multiple Argument Calls

With multiple arguments, just pass them in one by one, e.g.

```
(((e e1) e2) e3)
```

For example

```
>>> pat 31 42 56
3038
```