

Starting GHC

The GHC interpreter can be started from the Unix command prompt % by simply typing ghci:

Assignment Project Exam Help

```
% ghci
```

<https://powcoder.com>

```
GHCI, version 7.4.1: http://www.haskell.org/ghc/ :? for help
```

```
Loading package ghc-prim ... linking ... done.
```

```
Loading package integer-gmp ... linking ... done.
```

```
Loading package base ... linking ... done.
```

```
Prelude>
```

Add WeChat powcoder

The GHCi prompt `>` means that the interpreter is ready to evaluate an expression.

For example:

Assignment Project Exam Help

```
> 2+3*4  
14
```

Add WeChat powcoder

```
> (2+3) * 4  
20
```

```
> sqrt (3^2 + 4^2)  
5.0
```

The Standard Prelude

Haskell comes with a large number of standard library functions. In addition to the familiar numeric functions such as `+` and `*`, the library also provides many useful functions on lists.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

□ Select the first element of a list:

```
> head [1,2,3,4,5]  
1
```

□ Remove the first element from a list:

```
> tail [1,2,3,4,5]  
[2,3,4,5]
```

□ Select the nth element of a list:

```
> [1,2,3,4,5] !! 2  
3
```

<https://powcoder.com>

Add WeChat powcoder

□ Select the first n elements of a list:

```
> take 3 [1,2,3,4,5]  
[1,2,3]
```

□ Remove the first n elements from a list:

```
> drop 3 [1,2,3,4,5]  
[4,5]
```

□ Calculate the length of a list:

```
> length [1,2,3,4,5]  
5
```

<https://powcoder.com>

Add WeChat powcoder

□ Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]  
15
```

□ Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]  
120
```

□ Append two lists:

```
> [1,2,3] ++ [4,5]  
[1,2,3,4,5]
```

<https://powcoder.com>

Add WeChat powcoder

□ Reverse a list:

```
> reverse [1,2,3,4,5]  
[5,4,3,2,1]
```

Function Application

In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

Assignment Project Exam Help

<https://powcoder.com>

$f(a, b) + c \cdot d$ Add WeChat powcoder

Apply the function f to a and b , and add the result to the product of c and d .

In Haskell, function application is denoted using space, and multiplication is denoted using `*`.

Assignment Project Exam Help

`f a b + c*d` <https://powcoder.com>

Add WeChat powcoder

As previously, but in Haskell syntax.

Moreover, function application is assumed to have higher priority than all other operators.

Assignment Project Exam Help

`f a + b` <https://powcoder.com>

Add WeChat powcoder

Means $(f\ a) + b$, rather than $f\ (a + b)$.

Examples

Mathematics

$f(x)$

$f(x, y)$

$f(g(x))$

$f(x, g(y))$

$f(x)g(y)$

Haskell

$f\ x$

$f\ x\ y$

$f\ (g\ x)$

$f\ x\ (g\ y)$

$f\ x\ * \ g\ y$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Haskell Scripts

- As well as the functions in the standard library, you can also define your own functions;

Assignment Project Exam Help

- New functions are defined within a script, a text file comprising a sequence of definitions;

<https://powcoder.com>
Add WeChat powcoder

- By convention, Haskell scripts usually have a .hs suffix on their filename. This is not mandatory, but is useful for identification purposes.

My First Script

When developing a Haskell script, it is useful to keep two windows open, one running an editor for the script, and the other running GHCi.

Assignment Project Exam Help

<https://powcoder.com>

Start an editor, type in the following two function definitions, and save the script as test.hs:

Add WeChat powcoder

```
double x      = x + x
```

```
quadruple x = double (double x)
```

Leaving the editor open, in another window start up GHCi with the new script:

```
% ghci test.hs
```

Assignment Project Exam Help

Now both the standard library and the file test.hs are loaded, and functions from both can be used:

<https://powcoder.com>
Add WeChat powcoder

```
> quadruple 10  
40
```

```
> take (double 2) [1,2,3,4,5,6]  
?
```

Leaving GHCi open, return to the editor, add the following two definitions, and resave:

```
factorial n = product [1..n]
```

```
average ns = sum ns `div` length ns
```

Assignment Project Exam Help

<https://powcoder.com>

Note:

Add WeChat powcoder

□ `div` is enclosed in back quotes, not forward;

□ `x `f` y` is just syntactic sugar for `f x y`.

GHCi does not automatically detect that the script has been changed, so a reload command must be executed before the new definitions can be used:

```
> :reload
Reading file "test.hs"
> factorial 10
3628800
> average [1,2,3,4,5]
3
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Naming Requirements

- Function and argument names must begin with a lower-case letter. For example:

Assignment Project Exam Help

myFun

<https://powcoder.com>

fun1 arg_2

x'

Add WeChat powcoder

- By convention, list arguments usually have an s suffix on their name. For example:

xs

ns

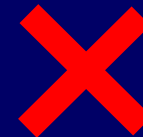
nss

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column:

Assignment Project Exam Help

a = 10	https://powcoder.com	a = 10
b = 20	Add WeChat powcoder	b = 20
c = 30		c = 30



The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.

```
a = b + c
  where
    b = 1
    c = 2
d = a * 2
```

<https://powcoder.com>

means

Add WeChat powcoder

```
a = b + c
  where
    {b = 1;
     c = 2}
d = a * 2
```

implicit grouping

explicit grouping