Haskell

System typów

W `ghci` za pomocą komendy interpretera `:type` (lub `:t`) można zapytać o typ wyrażenia.

```
Prelude> :t 'a'
'a' :: Char
Prelude> :t "ala"
"ala" :: [Char]
Prelude> :t "ala" ++ "ola"
"ala" ++ "ola" :: [Char]
Prelude> :t 2
2 :: (Num t) => t
Prelude> :t 2
2 :: (Num t) => t
Prelude> :t 2 + 2
2 + 2 :: (Num t) => t
Prelude> :t (+)
(+) :: (Num a) => a -> a -> a
Prelude> :t (+ 2)
(+ 2) :: (Num a) => a -> a
```

Haskell

Typy danych: Pary, n-tki

n-tka grupuje kilka wartości, które mogą być różnych typów

```
Prelude> (1,2)
(1,2)
Prelude> (1,"ala",'a')
(1,"ala",'a')
Prelude> fst (1,2)
1
Prelude> snd (1,2)
2
Prelude> :t ('a',"ala")
('a',"ala") :: (Char, [Char])
```

Haskell

Typy danych: listy

Lista jest ciągiem wartości jednolitego typu. Typ 'lista wartości typu t' zapisuje się [t].

```
Prelude> [1,2,3]
[1,2,3]
Prelude> ['a','b','c']          - napisy są listami znaków
"abc"
Prelude> ["Ala","Ola","Ula"]
["Ala","Ola","Ula"]
Prelude> :t ['a','b','c']
['a','b','c'] :: [Char]
Prelude> :t [1,2,3]
[1,2,3] :: (Num t) => [t]
```

Haskell

Typy danych: listy

Konstruktorem listy jest operator :

```
Prelude> 1:[]
[1]
Prelude> 1:2:3:4:5:[]
[1,2,3,4,5]
Prelude> :t (:)
(:) :: a -> [a] -> [a]
```

Haskell

Typy danych: listy

Podstawowe operacje na listach:

```
Prelude> head [1,2,3,4,5,6]
1
Prelude> tail [1,2,3,4,5,6]
[2,3,4]
Prelude> [1,2,3,4,5,6] !! 2
3
Prelude> take 4 [1,2,3,4,5,6]
[1,2,3,4]
```

Haskell

Typy polimorficzne

Typy polimorficzne to typy opisane z użyciem metazmiennych przebiegających zbiór typów, np.

`[a]`	lista wartości dowolnego typu
`a -> a`	funkcja zwracająca wynik tego samego typu co typ argumentu

Typy polimorficzne określają rodziny typów. Są narzędziem abstrakcji względem szczegółów nieistotnych w danym momencie. Funkcja wybierająca głowę listy działa tak samo, bez wzgledu na typ elementów listy. Ma ona typ `[a] -> a` .

Haskell

Klasy

Istnieje też w Haskellu pojęcie klasy. Klasa definiuje pewną własność, którą można potem nadać typom. Typy mogą być instancjami klas. Klasy można układać w hierarchie (dziedziczenie). Na przykład klasa Ord definuje własność porównywalności wartości należących do typu i wymaga od typów będących jej instancjami zdefiniowania następujących funkcji:

```
compare :: Ord a => a -> a -> Ordering
(<=)    :: Ord a => a -> a -> Bool
(<)     :: Ord a => a -> a -> Bool
(>=)    :: Ord a => a -> a -> Bool
(>)     :: Ord a => a -> a -> Bool
min     :: Ord a => a -> a -> a
max     :: Ord a => a -> a -> a
```