Tutorial

02 Variable & Type

Identifier Naming

Variables are examples of identifiers. *Identifiers* are names given to identify *something*.

There are some rules you have to follow for naming identifiers:

1. The first character of the identifier must be a letter of the alphabet (uppercase ASCII or lowercase ASCII or Unicode character) or an underscore (_).
2. The rest of the identifier name can consist of letters (uppercase ASCII or lowercase ASCII or Unicode character), underscores (_) or digits (0-9).
3. Identifier names are case-sensitive. For example, myname and myName are *not* the same. Note the lowercase n in the former and the uppercase N in the latter.

Examples of *valid* identifier names are i , name_2_3 . Examples of *invalid* identifier names are 2things , this is spaced out , my-name and >a1b2_c3 .

Keywords may be used only where the syntax permits, which can not be used as names.

Data Types

Variables can hold values of different types called *data types*. The basic types are numbers, strings, and our own types using class.

Type means:

1. size of the bytes occubied by the variable.
2. how to understand the bytes.
3. the meaning of the operations(such as, + - * /)

Static Typing, Dynamic Typing

Whether a language is statically typed or dynamically typed depends on how the compiler/interpreter handles variables and at what point during the execution of the program.

Static Typing

In a statically typed language, the variable itself has a type; if you have a variable that's an integer, you won't be able to assign any other type of value to it later. Some statically typed languages require you to write out the types of all your variables, while others will deduce many of them for you automatically.

Moreover, the compiler can tell which the type of a variable, without executing the program.

Dynamic Typing

In a dynamically typed language, a variable is simply a value bound to a name; the value(or object) has a type -- like "integer" or "string" or "list" -- but the variable itself doesn't. We can have a variable which, right now, holds a number, and later assign a string to it.

The programmer is free to bind names to different objects with a different type. So long as you only perform operations valid for the type the interpreter doesn't care what type they are.

```
#Python(Dynamic) OK
age = 12
age = "12"
```

```
//Go(static) A compile-time assignment error is raised.
age = 12
age = "12"
```

```
//Java(static) A compile-time assignment error is raised.
int age = 12;
age = "12"
```

Strong Typing and Weak Typing

The distinction between such typing comes to the fore when we write programming statements which involve operating values of different *types*.

Expected Behaviors

- **Strongly typed language:** more likely to generate an error (either at runtime or compile time depending on whether the language is interpreted or compiled).
- **Weakly typed language:** either produces unpredictable results or perform implicit type conversions to make sense of the expression.

Rule of Thumb: the more type coercions (implicit conversions) for built-in operators the language offers, the weaker the typing. (This could also be viewed as more built-in overloading of built-in operators.) i.e a strongly typed language is restrictive of type intermingling.

Python(Strong)

1. Adding a number to a string

```
>>> a, b = 10, 'K'
>>> a + b # Binary operation on different types
...
TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> a + ord(b) # Explicit type conversion of string to integer
85

>>> str(a) + b # Explicit type conversion of integer to string
'10K'
```

2. Adding an integer to a floating point number

```
>>> a, b = 1, 2.5
>>> a + b # No error
```

Conclusion: Raises errors when sense cannot be made. Also, the interpreter is not restrictive enough to spoil the fun in programming.

Go(Strong)

1. Adding a number to a string

```
func AddNumStr() {
     a, b := 10, "K"
     fmt.Println(a + b)
}
```

2. Adding an integer to a floating-point number

```
func AddNumFloat() {
     a, b := 1, 2.5
     fmt.Println(a + b)
}
```

Conclusion: In both the cases, an "invalid operation" compile-time error is raised.

Java(Strong)

1. Adding a number to a string

```
//3abc
int i=3;
System.out.println(3+"abc"); //compile it directly to "3abc"
System.out.println(i+"abc"); //makeConcatWithConstants to "3abc"
```

2. Adding an integer to a floating-point number

```
//6.0
int i=3;
System.out.println(3*2.0);//compile it directly to 6.0
System.out.println(i*2.0);// covert 3 to 3.0
```

3. divided by integer vs divided by double

```
int i =5;
System.out.println(i/2);  // 2
System.out.println(i/2.0);  // 2.5
```

Conclusion: In case 1&2, Java can get the expected results with complicated compiler behavior. In case 3, the compiler will check two value's type and decide which type of division will be called.

The Java programming language is also a strongly typed language, because types limit the values that a variable can hold or that an expression can produce, limit the operations supported on those values, and determine the meaning of the operations.

PHP(Weak)

```
$temp = “Hello World!”;
$temp = $temp + 10; // no error caused
echo $temp;
```

JavaScript(Weak)

1. Adding a number to a string

```
> a = 10
> b = "K"
> a + b
'10K'
```

1. Adding an integer to a floating-point number

```
> a = 1
> b = 2.5
> a + b
3.5
```

Conclusion: The compiler *implicitly* converts the type of values to make sense of the operation rather than raising errors.

Declarations

A declaration names a program entity and specifies some or all of its properties.

In Java:

1. A variable should be declared before it is called.
2. You have to declare the type of the variable.

In Go:

1. *variables* are explicitly declared and used by the compiler to e.g. check type-correctness of function calls.
2. `var` declares 1 or more variables.
3. You can declare multiple variables at once.
4. Go will infer the type of initialized variables.

In Python:

1. you don't have to declare what type each variable is.
2. variables are a storage placeholder for texts and numbers.
3. It must have a name so that you are able to find it again.
4. The variable is always assigned with the equal sign, followed by the value of the variable.