Ocaml Introduction

> OCaml is an industrial strength programming language supporting functional, imperative and object-oriented styles.

Functional Programming

Obviously, OCaml features functional programming while it still supports multiple paradigms. You can refer to OCaml official site's **introduction** of FP.

Functional programming is a programming paradigm where programs are constructed by applying and composing functions. Basically, it has following features.

Functions as first-class citizens

Function is a data type and functions can be assigned to variables, passed to other functions as parameters or returned by other functions.

Expression✅ Statement❌

Each step in functional programming is an expression that can be viewed as a pure calculation process that must has return value. While statements are more like description of operation, and have no return values.

This allows you to focus more on describing your problems and relationships between your data.

Pure Functions

A **pure function** is one without any **side-effects**. This means that given the same input, the functions will always return the same output, and they are not allowed to modify any values outside (based on the principle that variables in FP can never change once defined).

There are many advantages of using **pure functions**. One obvious advantage may be that you don't need to struggle with mutability of variables, scopes, memory issues and functions become easier to debug and maintain.

Lazy Evaluation

Lazy evaluation does not evaluate function arguments unless their values are required to evaluate the function call itself. This makes functions more like "real functions". A not so proper example:

```
print length([2+1, 3*2, 1/0, 5-4])
```

This line will fail in those languages using eager evaluation since `1/0` is illegal. While in functional programming, it will simply return 4, since the length of a list actually does not depend on its content.

This feature can speed up the program to some degree.

A Little Example

Both code snippets are Javascripts.

This also indicates that FP style is also preferred under some conditions in non-FP languages.

```
// Traditional Imperative Programming Style
const numList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
let result = 0;
for (let i = 0; i < numList.length; i++) {
  if (numList[i] % 2 === 0) {
    result += (numList[i] * 10)
  }
}

// Functional Programming Style
const result = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
               .filter(n => n % 2 === 0)
               .map(a => a * 10)
               .reduce((a, b) => a + b)
```

Actually the second form is more common in JS and python.

Installation

Refer to official documentations: **https://ocaml.org/docs/install.html**

OPAM: the package manager for OCaml.

Contents

Compilation

OCaml basically provides 4 ways to compile/run your code:

Interactice Mode

```
$ ocaml
        OCaml version 4.11.1

# 1+1;;
- : int = 2
```

OCaml provides an interactive toplevel, or REPL (Read–Eval–Print Loop) to evalute small expressions quickly.

The interactive command line of OCaml starts with `#`. Do not mix it with comments in shell scripts.

This approach is like what you will get if you type `python` in your terminal. This allows you to see the **types** of each function clearly, but not so convenient for large project organization.

You can quit the REPL by pressing `ctrl+D`.

Script Mode

OCaml source code can be written in `.ml` files and run directly with the `ocaml` command.

```
echo 'let () = print_endline "Hello, World!"' > test.ml
ocaml test.ml
```

This is also similar to other interpreted languages like `python`. No extra files will be generated.

Byte Code

```
ocamlc -o test test.ml
./test
# Or
ocamlc -o test.o test.ml
ocamlrun test.o
```

OCaml allows you to compile your source code into **byte-code** using `ocamlc`. The OCaml byte-code is run by OCaml runtime system `ocamlrun`.

In other words, `ocamlc` only generate code that can be recognized by OCaml, not machine code. This compiling process is much like `Java`. It has the advantage that byte-code can be **run directly on different platforms** without compiling again. However, it may run slower than machine code.

> If you look at the compiled result, it is actually a script starting with:
>
> ```
> #!/usr/local/bin/ocamlrun
> ```
>
> Therefore, the two commands above are equivalent.

Native Code

```
ocamlopt -o test test.ml
./test
```

OCaml also allows you to compile the source code all the way down to **native-code** using `ocamlopt`. This tool will generate **a standalone executable.** This file cannot be run on multiple platforms and always have larger size. However, the program can be executed more efficiently.

This process is much like other compiled languages.

Reference

1. Wikipedia
2. OCaml Official Site