

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

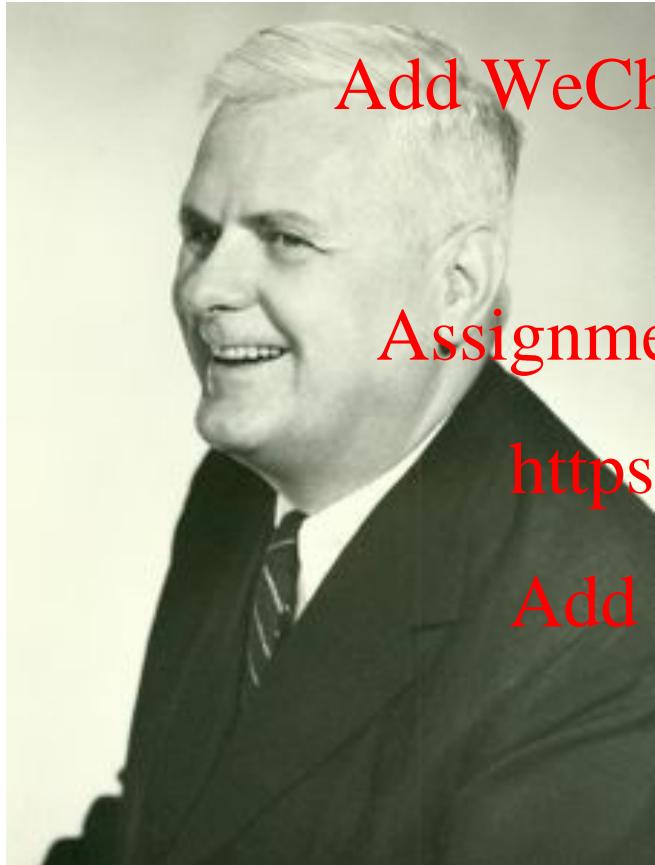
<https://powcoder.com>

Add WeChat powcoder

# INTRODUCTION TO OCAML

<https://powcoder.com>

## Assignment Project Exam Help



Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

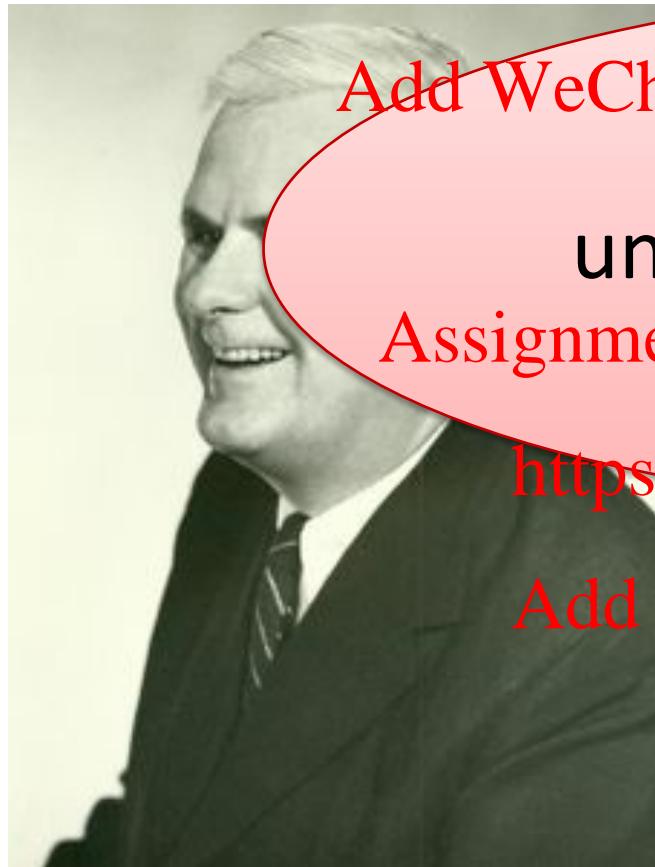
He said:  
Add WeChat powcoder

*"There may, indeed, be other applications of the system than its use as a logic."*

Alonzo Church, 1903-1995  
Princeton Professor, 1929-1967

<https://powcoder.com>

## Assignment Project Exam Help



Add WeChat powcoder  
Greatest technological  
understatement of the 20<sup>th</sup>  
Assignment Project Exam Help  
century?

<https://powcoder.com>

He said:  
Add WeChat powcoder

*"There may, indeed, be other  
applications of the system than  
its use as a logic."*

Alonzo Church, 1903-1995  
Princeton Professor, 1929-1967

# Vastly Abbreviated FP Genealogy

Assignment Project Exam Help

LCF Theorem

Prover (70s)

Add WeChat powcoder

Edinburgh ML

Assignment Project Exam Help

(80s-now)

Standard ML  
(90s-now)

Miranda (80s)

Haskell  
(90s - now)

Scala  
(00s - now)

F#  
(now)

OCaml  
(90s - now)

Coq  
(80s - now)

lazy

call-by-value

dependently typed

typed, polymorphic

# Vastly Abbreviated FP Genealogy

Assignment Project Exam Help

LCF Theorem  
Prover (70s)

Add WeChat powcoder

Edinburgh ML

Assignment Project Exam Help

Miranda (80s)

Standard ML  
(90s - now)

Haskell  
(90s - now)

Scala  
(00s - now)

OCaml  
(90s - now)

F#  
(now)

LISP  
(50s-now)

Scheme  
(70s-now)

Racket  
(00s-now)

untyped

Coq  
(80s - now)

lazy

call-by-value

dependently typed

typed, polymorphic

# Functional Languages: Who's using them?



- Elements of functional programming are showing up all over
  - F# in Microsoft Visual Studio
  - Scala combines ML (a functional language) with Objects
    - runs on the JVM
  - C# includes “delegates”
    - delegates == functions
  - Python includes “lambdas”
    - lambdas == more functions
  - Javascript
    - find tutorials online about using functional programming techniques to write more elegant code
  - C++ libraries for map-reduce
    - enabled functional parallelism at Google
  - Java has generics and GC
  - ...

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help  
**A Functional Review**

<https://powcoder.com>

Add WeChat powcoder

In Java or C, you get (most) work done by *changing* something

## Add WeChat powcoder

```
temp = pair.x;  
pair.x = pair.y;  
pair.y = temp;
```

Assignment Project Exam Help commands *modify* or *change* an existing data structure (like pair)

<https://powcoder.com>

In ML, you get (most) work done by *producing something new*

```
let  
  (x,y) = pair  
in  
  (y,x)
```

you *analyze* existing data (like pair)  
and you *produce* new data (y,x)

## Assignment Project Exam Help

pure, functional code:

```
let (x,y) = pair in  
(y,x)
```

imperative code:

## Add WeChat powcoder

```
temp = pair.x;  
pair.x = pair.y;  
pair.y = temp;
```

## Assignment Project Exam Help

- *outputs are everything*
- *output is function of input*
- *data properties are stable*
- *repeatable*
- *parallelism apparent*
- *easier to test*
- *easier to compose*
- *outputs are irrelevant!*
- *output is not function of input*
- *data properties change*
- *unrepeatable*
- *parallelism hidden*
- *harder to test*
- *harder to compose*

Assignment Project Exam Help  
Small core based on the *lambda calculus*.

- Control is based on (recursive) functions.
- Instead of for-loops, while-loops, do-loops, iterators, etc.
  - can be defined as library functions
- Makes it easy to define semantics



Assignment Project Exam Help  
Supports *first-class*, *lexically-scoped*, *higher-order* procedures

- a.k.a. first-class functions or closures or lambdas
- **first-class**: functions are data values like any other data value
  - like numbers, they can be stored, defined anonymously, ...
- **lexically-scoped**: meaning of variables determined statically
- **higher-order**: functions as arguments and results
  - programs passed to programs; generated from programs

These features also found in Racket, Haskell, SML, F#, Clojure, ....

# <https://powcoder.com> OCaml Feature Overview

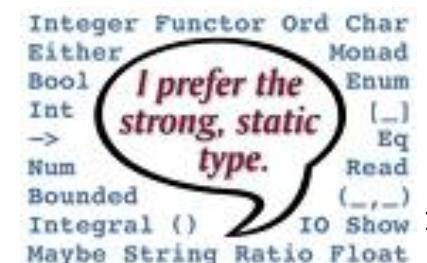
Statically typed: Assignment Project Exam Help

- compiler catches many silly errors before you can run the code
  - A type is worth a thousand tests (start at 6:20):
    - <https://www.youtube.com/watch?v=q1Yi-WM7XqQ>
- Java is also strongly, statically typed
- Python, Javascript, etc. are all strongly, *dynamically typed* – type errors are discovered while the code is running

Strongly typed: compiler enforces type abstraction

- cannot cast an integer to a record, function, string, etc.
- C/C++ are *weakly-typed* (statically typed) languages. The compiler will happily let you do something smart (*more often stupid*).

Type inference: compiler fills in types for you



- OCaml comes with compilers:
  - “ocamlc” – fast bytecode compiler
  - “ocamlopt” – optimizing, native code compiler
  - “ocamlbuild” – a nice wrapper that computes dependencies (not used in this course)
- And an interactive, top-level shell (interpreter):
  - useful for trying something out, and for debugging small programs
  - “ocaml” at the prompt
  - *You will likely use* <https://powcoder.com>
- And many other tools
  - e.g., debugger, dependency generator, profiler, etc.
- We will use it via VCL. You may want to try to install it yourself.
  - See OCaml.org

# <https://powcoder.com> Programs

- Many Options:
  - Emacs
    - what I'll be using in class, and what is available via VCL
    - good support for OCaml with the Tuareg Emacs Mode
    - powerful text editor, can run OCaml inside it
    - (extensions written in elisp – a functional language!)
  - OCaml IDE
    - integrated development environment written in OCaml
    - I haven't used it, so can't comment.
  - Eclipse
    - I haven't tried it but others recommend it. See the information on Brightspace added by TAs (in "Course Software" module, "Tips and Advanced Alternatives" submodule).
  - Sublime, atom
    - I haven't tried it, but many students have used it.

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

## **AN INTRODUCTORY EXAMPLE (OR TWO)**

# OCaml Compiler and Interpreter

- Demo
  - emacs    [Add WeChat powcoder](https://powcoder.com)
  - .ml files
  - writing simple programs: hello.ml, sumTo.ml
  - ocaml interpreter    [Assignment Project Exam Help](https://powcoder.com)
  - ocamlc compiler    <https://powcoder.com>

[Add WeChat powcoder](https://powcoder.com)

<https://powcoder.com> Program

Assignment Project Exam Help

Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n";;
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

## Assignment Project Exam Help

### Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

a function

<https://powcoder.com>

Add WeChat powcoder  
its string argument  
enclosed in "... "

no parens. normally call a function f like this:

f arg

(parens are used for grouping, precedence  
only when necessary)

a program  
can be nothing  
more than  
just a single  
expression  
(but that is  
uncommon)

# <https://powcoder.com> Program

## Assignment Project Exam Help

### Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

Assignment Project Exam Help

<https://powcoder.com>

interpreting and playing with hello.ml:

Add WeChat powcoder

the OCaml interpreter

```
$ ocaml
          OCaml Version 4.05.0
# 3 + 1;;
- : int = 4
#
```

Current versions of OCaml on website are 4.05 to 4.11 (depends on OS)

# <https://powcoder.com> Program

## Assignment Project Exam Help

### Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

**Assignment Project Exam Help**

<https://powcoder.com>

interpreting and playing with hello.ml:

**Add WeChat powcoder**

the OCaml  
interpreter

```
$ ocaml
          OCaml Version 4.05.0
# 3 + 1;;
- : int = 4
# #use "hello.ml";;
hello CSI 3120!!
- : unit = ()
#
```

Current  
versions of  
Ocaml on  
website are  
4.05 to 4.08  
(depends  
on OS)

# <https://powcoder.com> Program

## Assignment Project Exam Help

### Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

**Assignment Project Exam Help**

<https://powcoder.com>

interpreting and playing with hello.ml:

**Add WeChat powcoder**

the OCaml  
interpreter

```
$ ocaml
          OCaml Version 4.05.0
# 3 + 1;;
- : int = 4
# #use "hello.ml";;
hello CSI 3120!!
- : unit = ()
# #quit;;
$
```

Current  
versions of  
Ocaml on  
website are  
4.05 to 4.08  
(depends  
on OS)

## Assignment Project Exam Help

Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

Assignment Project Exam Help

<https://powcoder.com>

compiling and running hello.ml:

```
$ ocamlc hello.ml
```

Add WeChat powcoder

the OCaml compiler

## Assignment Project Exam Help

Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

Assignment Project Exam Help

<https://powcoder.com>

compiling and running hello.ml:

Add WeChat powcoder

```
$ ocamlc hello.ml
$ ocaml
          OCaml Version 4.05.0
#
```

the OCaml compiler

# <https://powcoder.com> Program

Assignment Project Exam Help

Add WeChat powcoder

hello.ml:

```
print_string "Hello CSI 3120!!\n"
```

Assignment Project Exam Help

<https://powcoder.com>

compiling and running hello.ml:

Add WeChat powcoder

```
$ ocamlc hello.ml
$ ocaml
          OCaml Version 4.05.0
# #load "hello.cmo";;
Hello CSI 3120!!
#
```

the OCaml compiler

# <https://powcoder.com> A Second OCaml Program

## Assignment Project Exam Help

Add WeChat powcoder

sumTo.ml:

```
(* sum the numbers from 0 to n
   Assignment Project Exam Help
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

a comment  
(\* ... \*)



# A Second OCaml Program

## Assignment Project Exam Help

Add WeChat powcoder  
the name of the function being defined

sumTo.ml:

```
(* sum the numbers from 0 to n
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

the keyword “let” begins a definition; keyword “rec” indicates recursion

# A Second OCaml Program

## Assignment Project Exam Help

### Add WeChat powcoder

sumTo.ml:

```
(* sum the numbers from 0 to n
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

Add WeChat powcoder

result type int

argument  
named n  
with type int

# <https://powcoder.com> A Second OCaml Program

## Assignment Project Exam Help

### Add WeChat powcoder

sumTo.ml:

```
(* sum the numbers from 0 to n
   Assignment Project Exam Help
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

a conditional statement in OCaml

# A Second OCaml Program

## Assignment Project Exam Help

Each branch of the conditional statement constructs a result

## Add WeChat powcoder

sumTo.ml:

```
(* sum the numbers from 0 to n
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

construct  
the result 0

construct  
a result  
using a  
recursive  
call to sumTo

# A Second OCaml Program

## Assignment Project Exam Help

### Add WeChat powcoder

sumTo.ml:

```
(* sum the numbers from 0 to n
   Assignment Project Exam Help
*)
let rec sumTo(n:int):int =
  if n <= 0 then
    0
  else
    n + sumTo (n-1)
let _ =
  print_int (sumTo 8);
  print_newline()
```

print the  
result of  
calling  
sumTo on 8

print a  
new line 30

# A Second OCaml Program (New Version)

## Assignment Project Exam Help

Add WeChat powcoder

newSumTo.ml:

```
(* sum the numbers from 0 to n
   precondition: n must be a natural number
*)
let rec sumTo(n:int):int =
  match n with
    0 -> 0
    | n -> n + sumTo (n-1)

let _ =
  print_int (sumTo 8);
  print_newline()
```

Additional comment

Assignment Project Exam Help

Add WeChat powcoder

# A Second OCaml Program (New Version)

## Assignment Project Exam Help

deconstruct the value n

## Add WeChat powcoder

newSumTo.ml:

```
(* sum the numbers from 0 to n
   precondition: n must be a natural number
*)
let rec sumTo(n:int):int =
  match n with
    0 -> 0
  | n -> n + sumTo (n-1)

let _ =
  print_int (sumTo 8);
  print_newline()
```

## Assignment Project Exam Help

<https://powcoder.com>

## Add WeChat powcoder

data to be  
deconstructed  
appears  
between  
key words  
“match” and  
“with”

# A Second OCaml Program

## Assignment Project Exam Help

vertical bar "|" separates the alternative patterns

Add WeChat powcoder

newSumTo.ml:

```
(* sum the numbers from 0 to n
   precondition: n must be a natural number
*)
let rec sumTo(n:int):int =
  match n with
    0 -> 0
  | n -> n + sumTo (n-1)

let _ =
  print_int (sumTo 8);
  print_newline()
```

deconstructed data matches one of 2 cases:

- (i) the data matches the pattern 0, or (ii) the data matches the variable pattern n

# A Second OCaml Program

## Assignment Project Exam Help

Each branch of the match statement constructs a result

### Add WeChat powcoder

newSumTo.ml:

```
(* sum the numbers from 0 to n
   precondition: n must be a natural number
*)
let rec sumTo(n:int):int =
  match n with
    0 -> 0
  | n -> n + sumTo (n-1)

let _ =
  print_int (sumTo 8);
  print_newline()
```

### Assignment Project Exam Help

### Add WeChat powcoder

as before,  
construct  
the result 0

as before,  
construct  
a result  
using a  
recursive  
call to sumTo

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# **OCAML BASICS: EXPRESSIONS, VALUES, SIMPLE TYPES**

# Terminology: Expressions, Values, Types

- **Expressions** are computations
  - $2 + 3$  is a computation
- **Values** are the results of computations
  - $5$  is a value
- **Types** describe collections of values and the computations that generate those values
  - `int` is a type
  - values of type `int` include
    - $0, 1, 2, 3, \dots, \text{max\_int}$
    - $-1, -2, \dots, \text{min\_int}$

# Some simple types, values, expressions

## Assignment Project Exam Help

Type:	Values:	Expressions:
int	-2, 0, 42 Add WeChat powcoder	42 * (13 + 1)
float	3.14, -1., 2e12	(3.14 +. 12.0) *. 10e6
char	'a', 'b', '&' Assignment Project Exam Help	int_of_char 'a'
string	"meow" "now"	"meow" ^ "now"
bool	true, false	if true then 3 else 4
unit	()	print_int 3 <a href="https://powcoder.com">https://powcoder.com</a>

Add WeChat powcoder

For more primitive types and functions over them,  
see the OCaml Reference Manual here:

<https://ocaml.org/releases/4.11/htmlman/libref/Stdlib.html>

# <https://powcoder.com> has a value

Assignment Project Exam Help

Expression:

42 * (13 + 1)	Add WeChat powcoder evaluates to 588
(3.14 +. 12.0) *. 10e6	↪ 151400000.
int_of_char 'a'	↪ 97
"moo" ^ "cow"	Assignment Project Exam Help ↪ "mooocow"
if true then 3 else 4	↪ 3
print_int 3	https://powcoder.com ↪ ()
1 + "hello"	Add WeChat powcoder does not evaluate!

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# **OCAML BASICS: CORE EXPRESSION SYNTAX**

# <https://powcoder.com> Core Expression Syntax

The simplest OCaml expressions  $e$  are:

- values      Add WeChat *numbers, strings, bools, ...*
- id              *variables (x, foo, ...)*
- $e_1 \text{ op } e_2$       *operators (x+3, ...)*
- $\text{id } e_1 \text{ } e_2 \dots e_n$       Assignment Project Exam Help  
*function call (foo 3 4 2)*
- **let**  $\text{id} = e_1$  **in**  $e_2$       *local variable decl.*
- **if**  $e_1$  **then**  $e_2$  **else**  $e_3$       *a conditional*  
Add WeChat *powcoder*
- $(e)$               *a parenthesized expression*
- $(e : t)$               *an expression with its type*

## <https://powcoder.com> A note on parentheses

In most languages, arguments are parenthesized & separated by commas:

`f (x, y, z)`   Add WeChat [powcoder](#)  
sum (3, 4, 5)

In OCaml, we don't write the parentheses or the commas:

`f x y z`   [Assignment](#) [Project](#) [Exam](#) [Help](#)  
sum 3 4 5

<https://powcoder.com>

But we do have to worry about *grouping*. For example,

Add WeChat [powcoder](#)

`f x y z`  
`f x (y z)`

The first one passes three arguments to `f` (`x`, `y`, and `z`)

The second passes two arguments to `f` (`x`, and the result of applying the function `y` to `z`.)