

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Concepts in Object-Oriented  
Assignment Project Exam Help  
Languages

<https://powcoder.com>

Add WeChat powcoder  
Mitchell Chapter 10

# Assignment Project Exam Help History

- Simula Add WeChat powcoder 1960's
    - Object concept used in simulation
  - Smalltalk 1970's
    - Object-oriented design, systems
  - C++ 1980's
    - Adapted Simula ideas to C
  - Java Add WeChat powcoder 1990's
    - Distributed programming, internet

## Assignment Project Exam Help Languages

- class-based languages
  - behavior of object determined by its class (in Mitchell Chapter 10)
- object-based
  - objects defined directly (can be done in OCaml)
- multi-methods
  - method to execute chosen based on characteristics of the arguments

# Assignment Project Exam Help Objects

- An object consists of

- hidden data

- instance variables, also called  
member data

- hidden functions also possible

- public operations

- methods or member functions

- can also have public variables

- in some languages

hidden data	
msg <sub>1</sub>	method <sub>1</sub>
⋮	⋮
msg <sub>n</sub>	method <sub>n</sub>

- Object-oriented program:
  - send messages to objects

# Assignment Project Exam Help Object-Oriented Languages

Add WeChat powcoder

- An *object-oriented language* refers to (in this chapter) a programming language that has objects and the following four features (concepts):
  - dynamic lookup
  - encapsulation (abstraction)
  - inheritance
  - subtyping
- Other topics
  - Inheritance is not Subtyping
  - A Comparison: Closures as Objects
  - A Comparison: Function-Oriented Program Organization

## Assignment Project Exam Help Dynamic Lookup

- *Dynamic lookup* means that when a message is sent to an object, the function code (or *method*) to be executed is determined by the way that the object is implemented
- Not determined by a static property of the pointer or variable used to name the object.
- In a sense, the object “chooses” how to respond to a message.
- Different objects may respond to the same message in different ways.

## Assignment Project Exam Help Dynamic Lookup

- In object-oriented programming,  
object → message (arguments)  
code depends on object and message

## Assignment Project Exam Help

- In conventional programming,  
operation (operands)  
meaning of operation is always the same

Fundamental difference between abstract data types and objects

## Assignment Project Exam Help

- Add two numbers ~~Add WeChat powcoder~~  $x + y$   
different **add** if **x** is integer, complex  
depends on the implementation of **x**
- Conventional programming ~~Assignment Project Exam Help~~  $\text{add}(x, y)$   
function **add** has fixed meaning, or  
different meanings depending on types (overloading), not on  
implementation ~~Add WeChat powcoder~~

Very important distinction:

Overloading is resolved at compile time.

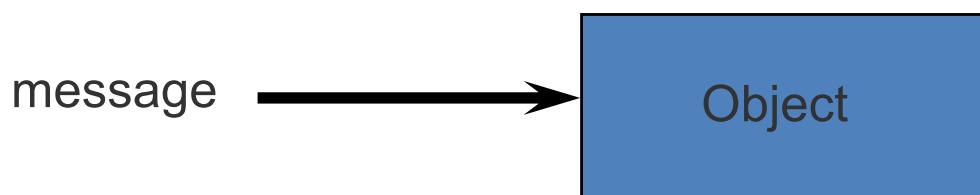
Dynamic lookup is resolved at run time.

(Dynamic lookup can be thought of as a run-time form  
of overloading.)

## Assignment Project Exam Help Encapsulation

- Builder of a concept has detailed view
- User of a concept has “abstract” view
- Encapsulation separates these two views
  - Implementation code: operate on representation
  - Client code: operate by applying fixed set of operations provided by <https://powcoder.com>

Add WeChat powcoder



## Assignment Project Exam Help Data Types

### Add WeChat powcoder

```
module type MATRIX =  
sig  
  type t  
  val create : unit -> t  
  val update : t -> int -> int -> unit  
  val add : t -> t -> t  
end
```

```
module Matrix : MATRIX =  
  struct  
    type t = ...  
    let create (...) : t = ...  
    let update (i:int) (j:int) (x:int) = ...  
      ... set m(i,j) to x ...  
    let add (m1:t) (m2:t) = ...  
  end
```

Add WeChat powcoder

```
class matrix  
... (representation)  
update(i,j,x) = ...set(i,j) of *this* matrix  
add(m) = ...add m to *this* matrix  
end
```

add(x,y) vs. x → add(y)

# Assignment Project Exam Help Comparison with Abstract Data Types

# Add WeChat powcoder

```
module type MATRIX =  
sig  
  type t  
  val create : unit -> t  
  val update : t -> int -> int  
  val add : t -> t -> t  
end  
  
class matrix  
  ... (r  
  upda  
  add(m)  
end
```

Both `x` and `y` must be matrices.  
If `add` were defined for complex numbers, integers, ..., the definition has to be either (hole) or static overloading, which requires a type constraint.

```
module Matrix : MATRIX =  
  struct  
    type t = ...  
    let c  
    [  
      The operation appears  
      to be missing  
    ]  
    int) (x:int) =
```

The operation appears to have only one argument—the matrix  $x$  that is receiving the message  $\text{add}(y)$ .

$\text{add}(x,y)$  vs.  $x \rightarrow \text{add}(y)$

Both  $x$  and  $y$  must be matrices.

If **add** were defined for complex numbers or integers, ..., then one definition hides the other (hole in scope) or static overloading required

## Assignment Project Exam Help Data Types

- Encapsulation via abstract data types
- Similarities with Objects (Shared Advantages)
  - Both combine functions and data
  - Both distinguish between a public interface and a private implementation
- Disadvantage as Compared to Object-Oriented Programming
  - Not extensible in the same way (see the next example)

## Assignment Project Exam Help Abstraction Example

```
module type INT_QUEUE = sig
  type t
  val mk_Queue : unit -> t
  val is_empty : t -> bool
  val add : int -> t
  val first : t -> int
  val rest : t -> t
  val length : t -> int
end
```

## Assignment Project Exam Help Abstraction Example: Regular Queue

```
exception Empty  
Add WeChat powcoder  
module IntQueue : INT_QUEUE =  
  struct  
    type t = int list  
    let mk_Queue () : t = []  
    let is_empty (q:t) = match q with | [] -> true  
                                      | _ -> false  
    let add (x:int) (q:t) : t = q @ [x]  
    let first (q:t) : int = match q with | [] -> raise Empty  
                                         | x::l -> x  
    let rest (q:t) : t = match q with | [] -> raise Empty  
                                         | _::l -> l  
    let rec length (q:t) : int = match q with | [] -> 0  
                                              | _::l -> 1 + length l  
  end
```

## Assignment Project Exam Help Abstraction Example: Priority Queues

```
module IntPriorityQueue WeChat powcoder
  struct
    type t = int list
    let mk_Queue () : t = []
    let is_empty (q:t) = match q with | [] -> true
    | _:_ -> false
    let rec add (x:int) (q:t) : t = match q with | [] -> [x]
    | y::l -> if x > y then x::y::l
    else y::add x l
    let first (q:t) : int = match q with | [] -> raise Empty
    | x::_ -> x
    let rest (q:t) : t = match q with | [] -> raise Empty
    | _::l -> l
    let rec length (q:t) : int = match q with | [] -> 0
    | _::l -> 1 + length l
  end
```

## Assignment Project Exam Help Comparison with Object-Oriented

- Abstract data types guarantee invariants of data structure
  - Client code does not have access to the internal representation of data
- Limited “reuse”
  - Both Queue and PQueue have the same interface
    - same number of operators, same names of operations, same types of operations
  - Both have same implementation except for “add”
  - Cannot apply Queue code to PQueue data, even though signatures are identical
  - Cannot form list of Queues and PQueues
- Implementations can be reused in object-oriented programming
  - In queue example, 5 functions have same implementation, can use inheritance to define one from the other.
- Data abstraction is an important part of OOP, the main innovation is that it occurs in an *extensible* form

## Assignment Project Exam Help

Consider a hospital system with several wait queues

- The billing department's queue is first-come, first-serve
- The emergency room's queue is a priority queue, based on severity of injury or illness.
- It would be useful to treat them uniformly, for example, to calculate the total number of people currently waiting.
- Can't apply PQueue length to a Queue or vice versa

Add WeChat powcoder

## Assignment Project Exam Help Subtyping and Inheritance

- Interface
  - The external view of an object
- Subtyping
  - Relation between interfaces
- Implementation
  - The internal representation of an object
- Inheritance
  - Relation between implementations

## Assignment Project Exam Help Subtyping in General

- In most typed languages (without subtyping), the application of a function  $f$  to an argument  $x$  requires some relation between the type of  $f$  and the type of  $x$ .
  - Common case:  $f$  is a function of type  $A \rightarrow B$  and  $x$  is an expression of type  $A$ . In the implementation of the type checker, find a type  $A \rightarrow B$  for  $f$ , find a type  $C$  for  $x$ , and check for equality:  $A = C$ .
- In languages with subtyping, subtyping is a relation on types based on *substitutivity*: if  $C$  is a subtype of  $A$  (written  $C <: A$ ), then any expression of type  $C$  may be used without type error in any context that requires an expression of type  $A$ .
  - Subtyping case: Find a type  $A \rightarrow B$  for  $f$ , find a type  $C$  for  $x$ , and check that  $C <: A$ .

## Assignment Project Exam Help Subtyping in General

- In most typed languages (without subtyping), the application of a function  $f$  to an argument  $x$  requires some relation between the type of  $f$  and the type of  $x$ .
  - Common case:  $f$  is a function of type  $A \rightarrow B$ ,  $x$  is an expression of type  $A$ . In the implementation checker, find a type  $A \rightarrow B$  for  $f$  and a type  $C$  for  $x$  such that  $C$  can be used in any context that requires an expression of type  $B$ . Example:  $A$  is `int` and  $C$  is `nat` (natural numbers which include 0 and positive integers).
- In languages with subtyping, types based on substitutivity (written  $C <: A$ ), then any expression of type  $C$  may be used without type error in any context that requires an expression of type  $A$ .
  - Subtyping case: Find a type  $A \rightarrow B$  for  $f$ , find a type  $C$  for  $x$ , and check that  $C <: A$ .

## Assignment Project Exam Help Advantages

- Permits uniform operations over various types of data
  - Makes it possible, for example, to have heterogeneous data structures that contain data that belong to different subtypes of some common type
  - Example:
    - A queue containing various bank accounts to be balanced. The “balance” operation is on bank accounts, which has subtypes such as saving, chequing, investment, etc.
    - A queue of type `bank_account` can contain all types of accounts.
- In object-oriented languages:
  - allows functionality to be added for subclasses without modifying the general parts of a system defined in the general class.

## Assignment Project Exam Help Object Interfaces

- Interface Add WeChat powcoder
  - The messages understood by an object
- Example: point
  - **x-coord** : returns x-coordinate of a point
  - **y-coord** : returns y-coordinate of a point
  - **move** : method for changing location
- The interface of an object is its type.

## Assignment Project Exam Help Subtyping

- If interface **B** contains all of interface **A**, then **B** objects can also be used as **A** objects.

Point

x-coord

y-coord

move

Coloured\_point

x-coord

y-coord

move

change\_colour

- Coloured\_point interface contains Point
  - Coloured\_point is a *subtype* of Point

## Assignment Project Exam Help Inheritance

- Implementation mechanism [Add WeChat powcoder](https://powcoder.com)
- New objects may be defined by reusing implementations of other objects
- Advantage: saves the effort of duplicating (or reading duplicated) [Assignment Project Exam Help](https://powcoder.com)
  - When one class is implemented by inheriting from another, changes to one affect the other
  - This has significant impact on code maintenance and modification
    - Only need to change in one place
    - Must be aware of what classes are affected by the change (all those that inherit the code)

## Assignment Project Exam Help

class Point

private

float x, y

public

point move (float dx, float dy);

## Add WeChat powcoder Subtyping

- Coloured points can be used in place of points
- Property used by client program

class Coloured\_point

<https://powcoder.com>

private

float x, y

colour c

public

point move (float dx, float dy);

point change\_colour (colour newc);

## Add WeChat powcoder Inheritance

- Coloured points can be implemented by reusing Point implementation
- Property used by implementer of classes

## Assignment Project Exam Help OO Program Structure

- Group together relevant data and functions
- Class
  - Defines behavior of all objects that are instances of the class
- Subtyping
- Inheritance

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Assignment Project Exam Help  
**RealWorld Ocaml**

<https://powcoder.com>

Add WeChat powcoder  
Chapters 11 and 12

# Assignment Project Exam Help Five Fundamental Properties of OO Programming

In Mitchell, Chapter 10, 4 features/concepts are described. Real World OCaml includes a fifth. Quoting:

- **Abstraction**

The details of the implementation are hidden in the object, and the external interface is just the set of publicly accessible methods.

- **Dynamic lookup**

When a message is sent to an object, the method to be executed is determined by the implementation of the object, not by some static property of the program. In other words, different objects may react to the same message in different ways.

- **Subtyping**

If an object **a** has all the functionality of an object **b**, then we may use **a** in any context where **b** is expected.

# Five Fundamental Properties of OO Programming

- Inheritance

The definition of one kind of object can be reused to produce a new kind of object. This new definition can override some behavior, but also share code with its parent.

- Open recursion

[Assignment Project Exam Help](https://powcoder.com)

An object's methods can invoke another method in the same object using a special variable (often called `self` or `this`). When objects are created from classes, these calls use dynamic lookup, allowing a method defined in one class to invoke methods defined in another class that inherits from the first.

new

# Assignment Project Exam Help Five Fundamental Properties of OO Programming

- Inheritance

The definition of one kind of object can be reused to produce a new kind of object. This new definition can override some behavior, but also share code with its parent.

- Open recursion

An object's methods can refer to another object using a special `new` keyword. New objects are created from existing objects, allowing a method defined in one class to be used in another class that inherits from it.

OCaml allows programming with objects without classes. It also includes classes, which are required for inheritance.

“Almost every notable modern programming language has been influenced by OOP, and you’ll have run across these terms if you’ve ever used C++, Java, C#, Ruby, Python, or JavaScript.”

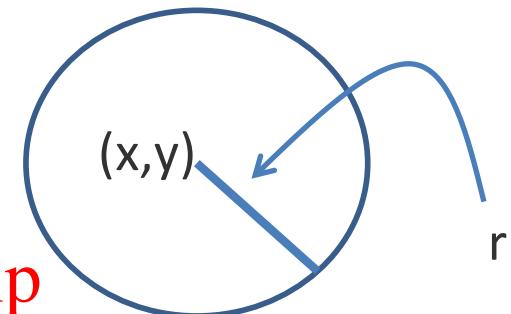
## Assignment Project Exam Help for Shapes

Add WeChat powcoder

```
type point = float * float  
type radius = float  
type side = float
```

```
type shape =  
| Circle of point * radius  
| Square of point * side  
| Ellipse of point * radius * radius  
| RtTriangle of point * side * side  
| Polygon of point * point list
```

Circle ( $p, r$ ) =

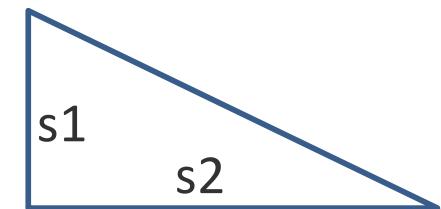


Add WeChat powcoder

Square ( $p, s$ ) =   
( $x, y$ ) } s

A diagram of a square centered at a point labeled (x, y). The side length is labeled s. A bracket indicates the side length from the center to one corner.

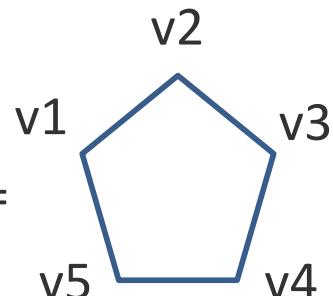
RtTriangle ( $p, s1, s2$ ) =



Ellipse ( $p, r1, r2$ ) =   
r1 r2

A diagram of an ellipse centered at a point labeled (x, y). The horizontal radius is labeled r1 and the vertical radius is labeled r2.

Polygon ( $p, [v1; ...; v5]$ ) =



## Assignment Project Exam Help Calculating Area

Add WeChat powcoder

```
type point = float * float
type radius = float
type side = float

type shape =
| Circle of point * radius
| Square of point * side
| Ellipse of point * radius * radius
| RtTriangle of point * side * side
| Polygon of point list
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```
let area (s : shape) : float =
  match s with
  | Circle (_, r) -> pi *. r *. r
  | Square (_, s) -> s *. s
  | Ellipse (_, r1, r2) -> pi *. r1 *. r2
  | RtTriangle (_, s1, s2) -> s1 *. s2 /. 2.
  | Polygon (_, ps) -> poly_area ps
```

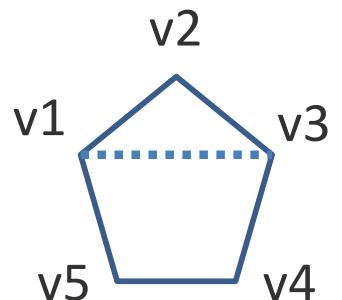
## Assignment Project Exam Help Area of a Polygon

Add WeChat powcoder

```
let tri_area (p1:point) (p2:point) (p3:point) : float =  
    let a = distance p1 p2 in  
    let b = distance p2 p3 in  
    let c = distance p3 p1 in  
    let s = 0.5 *. (a +. b +. c) in  
    sqrt (s *. (s -. a) *. (s -. b) *. (s -. c))
```

Assignment Project Exam Help

```
let rec poly_area (ps:point list):float =  
    match ps with  
    | p1 :: p2 :: p3 :: tail ->  
        tri_area p1 p2 p3 +. poly_area (p1 :: p3 :: tail)  
    | _ -> 0.
```



=



## An Object-Oriented Version

```
type point = float * float  
type radius = float  
type side = float  
  
class virtual shape = object  
    val mutable location : point = (0.0, 0.0)  
    method get_location = location  
    method set_location (xcoord:float) (ycoord:float) : unit =  
        location <- (xcoord, ycoord)  
    method virtual area: float  
end
```

# An Object-Oriented Version

```
type point = Add WeChat powcoder
type radius
type side = indicates an abstract class
class virtual shape = object
  val mutable location : float
  method get_location = location
  method set_location (xcoord, ycoord) : unit =
    location <- (xcoord, ycoord)
  method virtual read : float
end
```

Indicates an abstract method in an abstract class

<- used for assignment to mutable instance variables or fields

Unit () is not needed here. Keyword "method" indicates that, like a function, it must be called before executing the body.

Variables and methods are distinguished by the keywords "val" and "method"

## An Object-Oriented Version

```
type point = float * float
type radius = float
type side = float

class virtual shape = object
  val mutable location : point = (0.0,0.0)
  method get_location = location
  method set_location (xcoord:float) (ycoord:float) : unit =
    location <- (xcoord,ycoord)
  method virtual area : float
end

class circle = object
  inherit shape as super
  val mutable rad = 1.0
  method get_radius = rad
  method set_radius (r:radius) : unit = rad <- r
  method area = pi *. rad *. rad
end
```

# An Object-Oriented Version

```
type point = float * float
type radius = float
type side = float

class virtual shape = object
  val mutable location : point = (0.0, 0.0)
  method get_location : location = location
  method set_location (xcoord:float, ycoord:float) : unit =
    location <- (xcoord, ycoord)
  method area : float = 0.0
end

class circle = object
  inherit shape as super
  val mutable rad = 1.0
  method get_radius : radius = rad
  method set_radius (r:radius) : unit = rad <- r
  method area = pi *. rad *. rad
end
```

Inherits all instance variables and (non-abstract) methods

can add new ones

must implement abstract methods

Add WeChat powcoder

Add WeChat powcoder

# Assignment Project Exam Help Shape Classes and Objects

```
class circle = object
  inherit shape as super
  val mutable rad = 1.0
  method get_radius = rad
  method set_radius (r:radius) : unit = rad <- r
  method area = pi *. rad *. rad
end
```

## Assignment Project Exam Help

```
class square = object
  inherit shape as super
  val mutable side_length = 1.0
  method get_side = side_length
  method set_side (s:side) : unit = side_length <- s
  method area = side_length *. side_length
end
```

```
# let c1 = new circle;;
val c1 : circle = <obj>
# let s1 = new square;;
val s1 : square = <obj>
# let shape_1 : shape list = (c1 :> shape)::(s1 :> shape)::[];;
val shape_1 : shape list = [<obj>; <obj>]
```

# Assignment Project Exam Help Shape Classes and Objects

Add WeChat powcoder

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

keyword  
“new” for  
creating a  
new object

:>” is a type  
coercion operator;  
coercion only works  
if c1’s type is a  
subtype of shape

```
# let c1 = new circle;;
val c1 : circle = <obj>
# let s1 = new square;;
val s1 : square = <obj>
# let shape_1 : shape list = (c1 :> shape) :::(s1 :> shape) :::[];;
val shape_1 : shape list = [<obj>; <obj>]
```

## Assignment Project Exam Help Shape Processing Loop

```
let rec areas (sl : shape list) : float list =
  match sl with
  | [] -> []
  | h :: t -> (h#area) :: (areas t)

# let as1 = areas shape_1; Assignment Project Exam Help
val as1 : float list = [3.14159; 1.]
# c1#set_radius 2.0; https://powcoder.com
- : unit = ()
# s1#set_side 3.0; Add WeChat powcoder
- : unit = ()
# let as2 = areas shape_1;;
val as2 : float list = [12.56636; 9.]
```

Control “loop” or “iterator” does not know the type of each shape.

## Assignment Project Exam Help Objects (No Classes)

### Add WeChat powcoder

```
let st1 = object
    val mutable v = [0;2]
    method pop =
        match v with
        | [] -> None
        | h::t -> (v, Some h)
    method push h =
        v <- h::v
end
```

The type of an object is its interface, which is a collection of methods and their types.

```
val st1 : < pop : int option; push : int -> unit > = <obj>
# st1#pop;;
- : int option = Some 0
# st1#push 4;;
- : unit = ()
# st1#pop;;
- : int option = Some 4
```

### Add WeChat powcoder

## Assignment Project Exam Help Functions

```
let stack init : 'a option  
val mutable v = init  
method pop =  
  match v with  
  | [] -> None  
  | h::t -> (v <- t; Some h)  
method push h =  
  v <- h::v  
end
```

```
https://powcoder.com  
val stack : 'a list -> { pop : 'a option; push : 'a -> unit } =  
<fun>  
# let st2 = stack [3;2;1];;  
- : { pop : int option; push : int -> unit } = <obj>  
# st2#pop;;  
- : int option = Some 3
```

Finds the most general type for stack, which is polymorphic.

<https://powcoder.com>

Assignment Project Exam Help

Add WeChat powcoder

Concepts in Object-Oriented  
Assignment Project Exam Help  
Languages

<https://powcoder.com>

Add WeChat powcoder  
Mitchell Chapter 10 continued

## Inheritance and Abstraction

- 2 views of an abstraction in ordinary modules or abstract data types:
  - the client view
  - the implementation view
- 3 views with inheritance
  - the client view
  - the implementation view
  - the inheritance view (the view of classes that inherit from the class)
- Classes (object definitions) have 2 external clients
  - the *public interface* for the general users of the class (all the methods)
  - the *protected interface* for the inheritors (all the methods and all instance variables, i.e., data)

## Assignment Project Exam Help Subtyping Differs from Inheritance

Subtyping is a relation on interfaces, inheritance is a relation on implementations.

- Probably all class mechanisms that you are familiar with combine the two (e.g., C++, Java)
- An example that illustrates the difference (in languages that don't combine the two):
  - Queues: data structures with `Insert` and `delete` operations, such that the first element inserted is the first one removed (first-in, first-out)
  - Stacks: data structures with `insert` and `delete` operations, such that the first element inserted is the last one removed (last-in, first-out)
  - Dequeues (doubly-ended queue): Data structures with two `insert` and two `delete` operations. Allows insertion and deletion from each end.

## Inheritance vs. Subtyping Example

```
module type Add_WeChat = sig  
  type t  
  val mk_Dequeue : unit -> t  
  val insert_front : int -> t -> t  
  val insert_rear : int -> t -> t  
  val delete_front : int -> t -> t  
  val delete_rear : int -> t -> t  
  val first : t -> int  
  val last : t -> int  
  ...  
end
```

# Inheritance is Not Subtyping

## Implementation Add WeChat powcoder

- Implement `dequeue` first
- Then implement `stack` from `dequeue` by restricting access to `insert_rear` and `delete_rear`, e.g., override `insert_rear` to have the same implementation as `insert_front`
- And implement `queue` from `dequeue` by restricting access to `insert_front` and `delete_rear`
- `stack` and `queue` inherit from `dequeue`
- But `stack` and `queue` are not subtypes of `dequeue` (see next page)

## Inheritance is Not Subtyping

### Subtyping view Add WeChat powcoder

- Consider a function `f` that takes a `dequeue d` as an argument and then adds an element to both ends
- If `stack` or `queue` were a subtype of `dequeue`, then function `f` should work equally well when given a `stack s` or a `queue q`.
- However, the operation performed by `f` is not legal for either.
- In fact `dequeue` is a subtype of both `stack` and `queue`; any operation on a `stack` or `queue` would be a legal operation on a `dequeue`

## Assignment Project Exam Help Closures as Objects

### Object-Oriented Programming in Other Languages

- In a language with closures, we can simulate objects by using records (or tuples) that have function components.
  - Provides “dynamic lookup”
- If an object has private data (or functions), we can hide them with static scoping.
  - Provides a form of encapsulation

<https://powcoder.com>

Add WeChat powcoder

## Assignment Project Exam Help Example: Stacks as Closures

```
let create_stack : unit => WeChat powcoder
```

```
let store = ref [init] in
```

```
let push y = store := (y::!store) in
```

```
let pop () =
```

```
match !store with
```

```
| [] -> raise Assignment Project Exam Help
```

```
| h::t -> (store := t; h)
```

```
(push,pop)
```

Add WeChat powcoder

```
val create_stack : 'a -> ('a -> unit) * (unit -> 'a) = <fun>
```

## Assignment Project Exam Help Example: Stacks as Closures

```
val create_stack : unit -> (unit * int) = <fun>
```

```
# let (my_stk_push,my_stk_pop) = create_stack 0;;
```

```
val my_stk_push : int -> unit = <fun>
```

```
val my_stk_pop : unit -> int = <fun>
```

```
# my_stk_push 1;;
```

```
- : unit = ()
```

<https://powcoder.com>

```
# my_stk_pop();;
```

```
- : int = 1
```

```
# my_stk_pop();;
```

```
- : int = 0
```

```
# my_stk_pop();;
```

```
Exception: Empty.
```

## Assignment Project Exam Help Does this Work?

- Is “closure-oriented programming” the same as object-oriented programming?
- Provides:
  - dynamic lookup
  - encapsulation of private data
- But:
  - cannot substitute extended stacks for stacks (no subtyping)
  - only a weak form of inheritance
    - can add new operations to stack
    - not mutually recursive with old operations
- Object-oriented programming = “closure-oriented programming” + subtyping + inheritance

## A Comparison of OO and Function-Oriented Program Organization

Example

Add WeChat powcoder

- hospital simulation with data about doctors, nurses, and orderlies
- function or method to display information about an employee
- function or method to set or determine the pay of an employee

[Assignment Project Exam Help](https://powcoder.com)

<https://powcoder.com>

Add WeChat powcoder

## Assignment Project Exam Help Function-Oriented Organization

type nameAdd WeChat powcoder

type employee =

| Doctor of name \* ...

| Nurse of name \* ...

| Orderly of name \* Assignment Project Exam Help

let display (x:employee) = <https://powcoder.com> let pay (x:employee) =

match x with Add WeChat powcoder

| Doctor data -> ...

| Nurse data -> ...

| Orderly data -> ...

match x with

| Doctor data -> ...

| Nurse data -> ...

| Orderly data -> ...

## Assignment Project Exam Help Object-Oriented Organization

type nameAdd WeChat powcoder

class virtual employee =  
object

    method virtual display :  
unit

    method virtual display :  
end

<https://powcoder.com>

class doctor (n:name) =  
object

    inherit employee as super

    val mutable name = n

    val mutable salary = ...

    method display =...

    method pay =...

end

class nurse (n:name) =  
object...

class orderly (n:name) =  
object...

## Assignment Project Exam Help Summary

- Object-Oriented Languages have Objects plus the following 5 concepts
  - dynamic lookup
  - encapsulation (abstraction)
  - inheritance
  - subtyping
  - open recursion
- Other topics
  - Inheritance is not Subtyping
  - A Comparison: Closures as Objects
  - A Comparison: Function-Oriented Program Organization