

CS Bridge Module 15 Object Oriented Programming

1. Module 15

1.1 CS Bridge: Object-Oriented Programming Concepts

The screenshot shows a presentation slide with a purple header containing the NYU Tandon Online logo. The main content area has a light blue background with the title 'CS Bridge: Object-Oriented Programming Concepts' in green. Below the title, the author information 'Module 15' and 'Dan Katz' is displayed.

Notes:

1.2 In this module

The screenshot shows the 'In this module' section of the slide. It features a list of 15 bullet points describing various programming concepts. To the right of the list is a small video thumbnail showing a man in a striped shirt speaking.

- Definition of object and class
- The concept of encapsulation
- The creation of a class
- Enforcing protections and access
- Accessors/Mutators and const
- How to guarantee proper creation
- Using operators on classes
- Classes that contain dynamic memory
- Inheritance
- Polymorphism / Dynamic binding

1.3 Object orientation

Object orientation

NYU TANDON
ONLINE

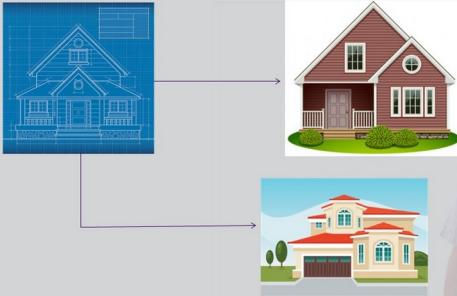
- A concept which came about in the early 1980s to bring together data and functions that operate on the data
- Creates, in code, an idea which exists in real life
- Protects the data from outside changes
- Allows for multiple operations to be performed with a single atomic function call
- Enforces the idea that the designer knows best about what to do, and not to do, to the data



1.4 Class vs. Object

Class vs. Object

NYU TANDON
ONLINE

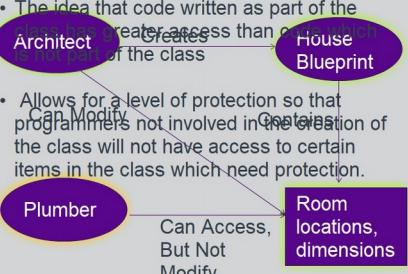


1.5 Encapsulation

Encapsulation

NYU TANDON
ONLINE

- The idea that code written as part of the class creates access to certain items which are not part of the class.
- Allows for a level of protection so that programmers not involved in the creation of the class will not have access to certain items in the class which need protection.



1.6 Creating a class

Creating a class

NYU TANDON
ONLINE

- A class is an abstract datatype
 - Used for creating a more complex datatype by combining simpler data types and adding capability
- In C++ the keyword “class” is used.
- For example, a “Date” in reality is three integers, a day, month and year.

```
class Date{  
public:  
    int day;  
    int month;  
    int year;  
};
```



1.7 Enforcing protection

Enforcing protection

NYU TANDON
ONLINE

- While the previous code showed a simplistic view of a class, we know that day is an integer which means its values could be -2.1 Billion to positive 2.1 billion! Not 1 to 31!
- Rather than allow uncontrolled (public) access to our data, we should protect our data as follows

(note) private is the DEFAULT for classes!

```
class Date{  
private:  
    int day;  
    int month;  
    int year;  
};
```



1.8 Accessors and Mutators

Accessors and Mutators

NYU TANDON
ONLINE

- Unfortunately, that doesn't work because now we can't access ANY of the variables.
- We need some functions to help us.
- Mutators can change data

```
class Date{  
private:  
    int day;  
    int month;  
    int year;  
public:  
    void setDate(int newDay);  
    void setMonth(int newMonth);  
    void setYear(int newYear)(year=newYear:);  
    void Date::setDay(int newDay){  
        if (newDay > 0 && newDay<=31)  
            day = newDay; }  
    void Date::setMonth(int newMonth){  
        if(newMonth >0 && newMonth<=12)  
            month=newMonth; }
```



1.9 Accessors and Mutators

Accessors and Mutators



- Now we can change the data, but not access it.
 - If we are not changing the data, we should warn C++ that we will not be making changes by marking the function as `const`.
 - `const` member functions cannot change data, and are the only functions that can be called if the object is `const`.
- ```
private:
 int day;
public:
 void setMonth(int newMonth);
 int getDay() const { return day; }
 int getMonth() const { return month; }
 int getYear() const { return year; }
};

void Date::setDay(int newDay)
{
 if (newDay > 0 && newDay <= 31)
 day = newDay;
}

void Date::setMonth(int newMonth)
{
 if(newMonth > 0 && newMonth <= 12)
 month=newMonth;
}
```



## 1.10 Other useful functions

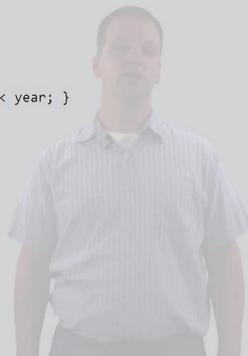
### Other useful functions



- Perhaps we could add a `printDate` function to print a formatted date

```
void displayDate() const{ cout << day << "/" << month << "/" << year; }
```

- Or a validate function to check if the date is valid
- The above might require a leap year calculation which could be a private function inside Date
- The possibilities are endless!



## 1.11 Creating and working with an object

### Creating and working with an object



- You can create an object just like any other datatype
- Working with an object requires the “dot operator”
- Remember, you will NOT have access to private!

```
int main(){
 Date d1;
 d1.setDay(6);
 d1.setMonth(8);
 d1.setYear(1991);

 cout << "Very important date: ";
 d1.displayDate();
```



## 1.12 Constructors

### Constructors



NYU TANDON  
ONLINE

- If we just create an object, its data members would be garbage values
- Object orientation means knowing that we will NEVER have invalid values, even if it was just created
- So C++ provides for constructors which are functions that are automatically called when the object is created
- The default constructor is a function named exactly the same as the name of the class (case sensitive) with no return type (not void, nothing) and no parameters.

## 1.13 Constructor parts

### Constructor parts



NYU TANDON  
ONLINE

```
class Date{
private:
 int day;
 int month;
 int year;
public:
 Date():day(1), month(1), year(1970){}

 Member initialization list

class Date{
private:
 int day;
 int month;
 int year; Either solution is fine, for now
public:
 Date() {
 day = 1;
 month = 1;
 year = 1970;
 }
}
```

## 1.14 More constructors

### More constructors



NYU TANDON  
ONLINE

- A constructor that takes three ints?  
`Date(int newD, int newM, int newY) :day(newD), month(newM), year(newY) {}`
- Now we can do  
`int main(){  
 Date d2(6, 8, 1991);  
}`
- Create as many constructors as you would like, each must have a unique set of parameters

## 1.15 An important pointer

An Important Pointer

NYU TANDON  
ONLINE

- Every object has a pointer, which looks like a data member, called “this”
- The “this” pointer points to the calling object
- Though not necessary, you can always use this

```
void setYear(int newYear) { this -> year = newYear; }
```

- Some times “this” will be needed
  - If a person has a spouse pointer, and gets married, the person’s spouse pointer should point to the new spouse but where does the spouse’s spouse pointer point? Answer: this



## 1.16 Operator Overloading

Operator Overloading

NYU TANDON  
ONLINE

|        |   |     |   |             |
|--------|---|-----|---|-------------|
| Coffee | + | Ice | → | Iced Coffee |
| Coffee | + | +   | → | 2 Coffees   |
| Coffee | + | =   | → | 3 Coffees   |

- Operators are just functions with a strange function call
  - a+b is actually a function call to either:
    - operator+(a,b)
    - a.operator+(b)
- Operator functions can be member or non-member
- Some operators change the data in an object, some return a new object



## 1.17 Operator restrictions

Operator Restrictions

NYU TANDON  
ONLINE

- Some operators cannot be overloaded
  - .
  - ::
  - \*
  - sizeof
  - ?:
- Some can only be overloaded as a member
  - = (the assignment operator)
  - [] (the array-index of operator)
- Some, almost, cannot be overloaded as a member
  - <<
  - >>



## 1.18 Knowledge Check

(Multiple Response, 10 points, 4 attempts permitted)

Knowledge Check

NYU TANDON  
ONLINE

Which of the following operators CANNOT be overloaded?  
(Select all that apply)

sizeof

+=

\*

\*.

==

?:

| Correct | Choice |
|---------|--------|
| X       | sizeof |
|         | +=     |
|         | *      |
| X       | *.     |
|         | ==     |
| X       | ?:     |

**Feedback when correct:**

That's right! You selected the correct response.

**Feedback when incorrect:**

You did not select the correct response.

## Correct (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

Which of the following operators CANNOT be overloaded?  
(Select all that apply)

Sizeof  
 +=  
 \*.  
 \*:  
 ==  
 ?:

Correct

That's right! You selected the correct response.

Continue

## Incorrect (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

Which of the following operators CANNOT be overloaded?  
(Select all that apply)

Sizeof  
 +=  
 \*.  
 \*:  
 ==  
 ?:

Incorrect

You did not select the correct response.

Continue

## Try Again (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

Which of the following operators CANNOT be overloaded?  
(Select all that apply)

Sizeof  
 +=  
 \*.  
 \*:  
 ==  
 ?:

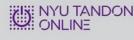
Incorrect

That is incorrect. Please try again.

Try Again

## 1.19 Choosing member vs. non-member

### Choosing Member vs. Non-Member



- Remember that members have access to private
- A function defined as a “friend” in the class is NOT a member, but will have access to private
- Otherwise, there is only one difference
  - $a+b$  will work in either case
  - $a+5$  (object + constant) will work if there is a constructor which can take 5 and construct an object to add to a
  - $5+a$  (constant + object) will only work if the above constructor exists AND operator+ is a non-member!



## 1.20 What to return?

### What To Return?



- The value returned depends on what the operator should do
- The return data type depends on what is being returned
  - If the item being returned was created inside the function, return MUST be by value
  - If the item was passed in as a parameter, or you are returning the “this” pointer, you can return by reference
- Returning by reference is preferred



## 1.21 An Odd Case

### An Odd Case



- Because C++ has both a pre- and post-increment operator, we need a special case to differentiate
  - The pre increment operator will change the value and then return a reference to the existing object  
`Date& operator++();`
  - The post-increment operator will copy the object, then change the value and return the copy. `Date operator++(int);`
  - Because we need to differentiate, the post-increment is passed an int.
  - The int passed has no bearing on the operator, its just a way to differentiate



## 1.22 Classes that contain dynamic memory

Classes that contain dynamic memory

NYU TANDON  
ONLINE

- All classes have an assignment operator and constructor which can copy an existing object, this is a leftover of C's structs and is useful `new int(newVal) {}`
- Unfortunately, when pointers are involved, ~~the built-in~~ operators copy the pointers not what the pointers are pointing to.
- This is known as a shallow copy and would be problematic if left alone

one = two

## 1.23 Three problem, Big 3 solution

Three problem, Big 3 solution

NYU TANDON  
ONLINE

- Copy operations need to copy the data, not the pointers `*value;`
- Since we are creating memory in the constructor, we need to destroy that memory when the object falls out of scope `~Thing() { delete value; } //destructor`
- The Big 3 are three functions that, if you need any of them, you need them all
  - Destructor – Called automatically when the object falls out of scope
  - Copy Constructor – Constructs an object based on an existing one
  - Assignment Operator – Copies one object to another (deep copy)
    - Be careful of avoiding `x=x` (self-assignment use: `"if(this==&rhs)"` to avoid)

## 1.24 Knowledge Check

(Matching Drag-and-Drop, 10 points, 3 attempts permitted)

## Knowledge Check



In the following, match the descriptions of the BIG 3.

|                     |                                                         |
|---------------------|---------------------------------------------------------|
| Destructor          | Called automatically when the object falls out of scope |
| Copy constructor    | Constructs an object based on an existing one           |
| Assignment operator | Copies one object to another (deep copy)                |

| Correct             | Choice                                                  |
|---------------------|---------------------------------------------------------|
| Destructor          | Called automatically when the object falls out of scope |
| Copy constructor    | Constructs an object based on an existing one           |
| Assignment operator | Copies one object to another (deep copy)                |

### Feedback when correct:

That's right! You selected the correct response.

### Feedback when incorrect:

You did not select the correct response.

### Notes:

## Correct (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

In the following, match the descriptions of the BIG 3.

Correct

That's right! You selected the correct response.

Destructor      Calls out of scope

Copy const      Existing one

Assignment operator      Copies one object to another (deep copy)

Continue

This screenshot shows a 'Knowledge Check' slide from NYU Tandon Online. The slide title is 'Knowledge Check' and the institution logo is 'NYU TANDON ONLINE'. The main text asks, 'In the following, match the descriptions of the BIG 3.' A callout box at the top center says 'Correct' and 'That's right! You selected the correct response.' Below it, three items are listed: 'Destructor' (Calls out of scope), 'Copy const' (Existing one), and 'Assignment operator' (Copies one object to another (deep copy)). A green 'Continue' button is visible at the bottom of the callout box.

## Incorrect (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

In the following, match the descriptions of the BIG 3.

Incorrect

You did not select the correct response.

Destructor      Calls out of scope

Copy const      Existing one

Assignment operator      Copies one object to another (deep copy)

Continue

This screenshot shows a 'Knowledge Check' slide from NYU Tandon Online. The slide title is 'Knowledge Check' and the institution logo is 'NYU TANDON ONLINE'. The main text asks, 'In the following, match the descriptions of the BIG 3.' A callout box at the top center says 'Incorrect' and 'You did not select the correct response.' Below it, three items are listed: 'Destructor' (Calls out of scope), 'Copy const' (Existing one), and 'Assignment operator' (Copies one object to another (deep copy)). A green 'Continue' button is visible at the bottom of the callout box.

## Try Again (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

In the following, match the descriptions of the BIG 3.

Incorrect

That is incorrect. Please try again.

Destructor      Calls out of scope

Copy const      Existing one

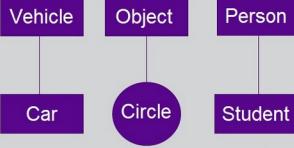
Assignment operator      Copies one object to another (deep copy)

Try Again

This screenshot shows a 'Knowledge Check' slide from NYU Tandon Online. The slide title is 'Knowledge Check' and the institution logo is 'NYU TANDON ONLINE'. The main text asks, 'In the following, match the descriptions of the BIG 3.' A callout box at the top center says 'Incorrect' and 'That is incorrect. Please try again.' Below it, three items are listed: 'Destructor' (Calls out of scope), 'Copy const' (Existing one), and 'Assignment operator' (Copies one object to another (deep copy)). A green 'Try Again' button is visible at the bottom of the callout box.

## 1.25 Inheritance

Inheritance

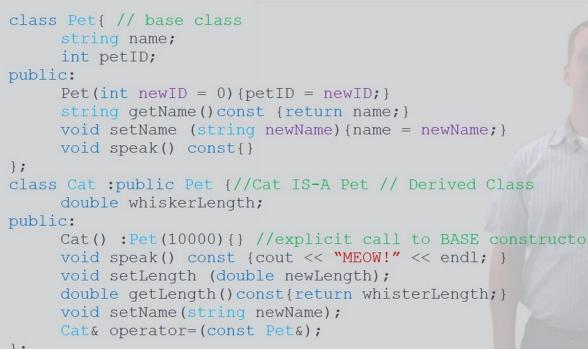


- During inheritance, the existing class, called a base class is enlarged to form a derived class
- All items (functions and data) which exist in the base class will be in the derived class automatically
- The derived class can create new versions of existing functions, this is called overriding
- This does not change private access, the derived class cannot access private!

NYU TANDON  
ONLINE

## 1.26 Pets and Cats

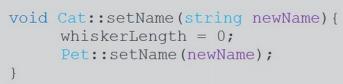
Pets and Cats



NYU TANDON  
ONLINE

## 1.27 What if we need to override?

What If We Need To Override?



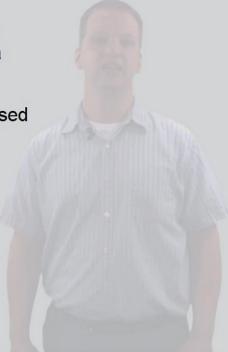
NYU TANDON  
ONLINE

## 1.28 What if derived **SHOULD** access base's stuff?

What if derived **SHOULD** access base's stuff?

NYU TANDON  
ONLINE

- C++ adds a protection mechanism to allow for a condition where a derived class should access a base member's item
- Items listed as "protected" in base can be accessed from derived
- This should be used sparingly



## 1.29 Knowledge Check

(Matching Drag-and-Drop, 10 points, 3 attempts permitted)

Knowledge Check

NYU TANDON  
ONLINE

List the differences between public/private/protected members.

|                   |                                                                                                |
|-------------------|------------------------------------------------------------------------------------------------|
| Private Members   | Can be accessed only from the class that it is a part of, or those that are marked as "friend" |
| Public Members    | Can be accessed from anywhere inside or outside the class                                      |
| Protected Members | Can be accessed only from the base class, or derived classes                                   |

| Correct           | Choice                                                                                         |
|-------------------|------------------------------------------------------------------------------------------------|
| Private Members   | Can be accessed only from the class that it is a part of, or those that are marked as "friend" |
| Public Members    | Can be accessed from anywhere inside or outside the class                                      |
| Protected Members | Can be accessed only from the base class, or derived classes                                   |

### **Feedback when correct:**

That's right! You selected the correct response.

### **Feedback when incorrect:**

You did not select the correct response.

### **Correct (Slide Layer)**

The screenshot shows a "Knowledge Check" slide titled "List the differences between public/private/protected members." A callout box labeled "Correct" contains the message "That's right! You selected the correct response." Below the slide content, there are three options: "Private Members" (disabled), "Public Members" (selected and highlighted in green), and "Protected Members". A "Continue" button is visible at the bottom right of the slide area.

### **Incorrect (Slide Layer)**

The screenshot shows a "Knowledge Check" slide titled "List the differences between public/private/protected members." A callout box labeled "Incorrect" contains the message "You did not select the correct response." Below the slide content, there are three options: "Private Members" (disabled), "Public Members" (selected and highlighted in green), and "Protected Members". A "Continue" button is visible at the bottom right of the slide area.

## Try Again (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

List the differences between public/private/protected members.

Incorrect

That is incorrect. Please try again.

Private Members

Public Members

Protected Members

Try Again

Can be accessed only from the base class, or derived classes

## 1.30 Polymorphism

Polymorphism

NYU TANDON  
ONLINE

- Polymorphism in C++ allows us to copy data from a derived class into a base class. But only the base items will copy over. (String)
- and pets
- Overloading the assignment operator can allow us to copy from a base class into a derived object. (overloading)
- Since every cat is a pet, every cat will contain all of the functions inside
- Since every derived object contains everything in base, a base pointer can point to a derived object. (polymorphism)
  - Though not necessarily the same versions
- (there is no guarantee that base contains everything in derived so a derived pointer can NEVER point to a base object)

```
int main(){
 Pet p;
 Pet* pptr;
 Cat c;
 Cat* cptr;
 p = c; // Always allowed
 c = p; // Allowed if operator=(const Pet&) is overloaded;
 pptr = &c; // Always allowed, polymorphism
 cptr = &p; // NEVER ALLOWED!
}
```

## 1.31 Virtual Functions

Virtual Functions

NYU TANDON  
ONLINE

- If a base pointer is used to point to a derived object, by default, the functions called will be the BASE versions of the functions.
  - This can be catastrophic if the derived function does something different than the base
- The solution is to mark the base function as "virtual"
  - This employs late (aka dynamic) binding
  - The version of the function called depends on the type of object, not the type of pointer

## 1.32 Pure virtual

Pure virtual

If the base class should contain a function, but doesn't know what the function should actually do, the function can be marked as pure virtual "=0"

```
class Pet //base class (abstract because of speak) Hover
 string name;
 int petID;
public:
 Pet(int newID = 0){petID = newID;}
 string getName() const {return name;}
 virtual void setName(string newName){name=newName;}
 virtual void speak() const =0; //Pure virtual function
};

class Cat :public Pet //Cat IS A Pet // Derived class
 double whiskerLength;
public:
 Cat() :Pet(10000){} //explicit call to BASE constructor
 void speak() const{cout<< "MEOW!" << endl; }
 void setLength(double newLength);
 double getLength() const{return whiskerLength; }
 void setName(string newName);
 Cat& operator=(const Pet&);

};
```



## 1.33 Knowledge Check

(Multiple Choice, 10 points, 4 attempts permitted)

Knowledge Check

In the following code, what will be printed if a = 12 and b = 6

```
class Polygon {
protected:
 int width, height;
public:
 void set_values (int a, int b)
 { width=a; height=b; }
 int area () const = 0
};
class Rectangle: public Polygon {
public:
 int area ()
 { return width * height; }
};
int main ()
{
 Rectangle rect;
 rect.set_values (a + 3,b * 2);
 cout << rect.area() << '\n';
 return 0;
}
```

180  
 432  
 210  
 288

| Correct | Choice |
|---------|--------|
| X       | 180    |
|         | 432    |
|         | 210    |
|         | 288    |

### **Feedback when correct:**

That's right! You selected the correct response.

### **Feedback when incorrect:**

You did not select the correct response.

## **Correct (Slide Layer)**

Knowledge Check 

In the following code, what will be printed if a = 12 and b = 6

**Correct**

That's right! You selected the correct response.

**Continue**

```
class Polygon {
protected:
 int width,
public:
 void set_val()
 { width=a;
 int area()
 };
class Rectangle
public:
 int area()
 { return width * height; }
};
int main () {
 Rectangle rect;
 rect.set_values (a + 3,b * 2);
 cout << rect.area() << '\n';
 return 0;
}
```

210  
 288

## **Incorrect (Slide Layer)**

Knowledge Check 

In the following code, what will be printed if a = 12 and b = 6

**Incorrect**

You did not select the correct response.

**Continue**

```
class Polygon {
protected:
 int width,
public:
 void set_val()
 { width=a;
 int area()
 };
class Rectangle
public:
 int area()
 { return width * height; }
};
int main () {
 Rectangle rect;
 rect.set_values (a + 3,b * 2);
 cout << rect.area() << '\n';
 return 0;
}
```

210  
 288

## Try Again (Slide Layer)

Knowledge Check

NYU TANDON  
ONLINE

In the following code, what will be printed if a = 12 and b = 6

```
class Polygon {
protected:
 int width,
public:
 void set_val()
 { width=a;
 int area ()
 };
class Rectangle
public:
 int area ()
 { return width * height; }
};
int main () {
 Rectangle rect;
 rect.set_values (a + 3,b * 2);
 cout << rect.area() << '\n';
 return 0;
}
```

Incorrect

That is incorrect. Please try again.

Try Again

210  
 288

## 1.34 The same function call results in different output

The same function call results in different output

NYU TANDON  
ONLINE

```
class Dog :public Pet{
 double earSize;
public:
 Dog() : Pet(20000){}
 void speak() const{ cout << "WOOF!" << endl; }
 void setSize (double newSize)(earSize = newSize);
 double getSize() const{ return earSize; }
};

int main(){
 Pet** pArray = new Pet*[3];
 pArray[0] = new Cat();
 pArray[1] = new Dog();
 pArray[2] = new Cat();
 for (int i=0; i<3; i++)
 pArray[i]->speak(); //MEOW, Woof, MEOW
}
```



## 1.35 What we discussed

What we discussed

NYU TANDON  
ONLINE

- Definition of object and class
- The concept of encapsulation
- The creation of a class
- Enforcing protections and access
- Accessors/Mutators and const
- How to guarantee proper creation
- Using operators on classes
- Classes that contain dynamic memory
- Inheritance
- Polymorphism / Dynamic binding



## 1.36 Results Slide

(Results Slide, 0 points, 1 attempt permitted)

The screenshot shows a results slide with the following details:

- Header: Results, NYU TANDON ONLINE
- Score: Your Score: %Results.ScorePercent%% (%Results.ScorePoints% points)
- Score: Passing Score: %Results.PassPercent%% (%Results.PassPoints% points)
- Text: Result:
- Buttons: Retry Quiz, Review Quiz

| Results for          |
|----------------------|
| 1.18 Knowledge Check |
| 1.24 Knowledge Check |
| 1.29 Knowledge Check |
| 1.33 Knowledge Check |

Result slide properties

Passing 80%

Score

**Notes:**

## Success (Slide Layer)

The slide layer for success displays the NYU Tandon Online logo at the top right. The main content area shows 'Your Score:' followed by a percentage and '(%Results.ScorePoints% points)', and 'Passing Score:' followed by another percentage and '(%Results.PassPoints% points)'. Below this, a 'Result:' section indicates a pass with a green checkmark and the message 'Congratulations, you passed.' At the bottom are two buttons: 'Retry Quiz' and 'Review Quiz'.

## Failure (Slide Layer)

The slide layer for failure displays the NYU Tandon Online logo at the top right. The main content area shows 'Your Score:' followed by a percentage and '(%Results.ScorePoints% points)', and 'Passing Score:' followed by another percentage and '(%Results.PassPoints% points)'. Below this, a 'Result:' section indicates a fail with a red X and the message 'You did not pass.' At the bottom are two buttons: 'Retry Quiz' and 'Review Quiz'.

### **1.37 End of Module**



End of Module

Exit