Première Programme en Objective-C

Exercice -1

```
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
- (void)sampleMethod;
@end

@implementation SampleClass

- (void)sampleMethod{
   NSLog(@"Hello, World! \n");
}

@end

int main()
{
   /* my first program in Objective-C */
   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass sampleMethod];
   return 0;
}
```

Fonctions

Exercice -2

Appeler par Valeur

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
/* method declaration */
- (void)swap:(int)num1 andNum2:(int)num2;
@end

@implementation SampleClass

- (void)swap:(int)num1 andNum2:(int)num2
{
   int temp;

   temp = num1; /* save the value of num1 */
   num1 = num2;    /* put num2 into num1 */
   num2 = temp; /* put temp into num2 */

}

@end

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   SampleClass *sampleClass = [[SampleClass alloc]init];

   NSLog(@"Before swap, value of a : %d\n", a );
   NSLog(@"Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   [sampleClass swap:a andNum2:b];

   NSLog(@"After swap, value of a : %d\n", a );
   NSLog(@"After swap, value of b : %d\n", b );

   return 0;
}
```

Exercice -3

Appeler par Reference

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
/* method declaration */
- (void)swap:(int *)num1 andNum2:(int *)num2;
@end

@implementation SampleClass

- (void)swap:(int *)num1 andNum2:(int *)num2
{
   int temp;

   temp = *num1; /* save the value of num1 */
   *num1 = *num2;    /* put num2 into num1 */
   *num2 = temp; /* put temp into num2 */

   return;

}

@end

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;

   SampleClass *sampleClass = [[SampleClass alloc]init];

   NSLog(@"Before swap, value of a : %d\n", a );
   NSLog(@"Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   [sampleClass swap:&a andNum2:&b];

   NSLog(@"After swap, value of a : %d\n", a );
   NSLog(@"After swap, value of b : %d\n", b );

   return 0;
}
```

Objective-C Numéros

Exercice -4

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject

- (NSNumber *)multiplyA:(NSNumber *)a withB:(NSNumber *)b;

@end

@implementation SampleClass

- (NSNumber *)multiplyA:(NSNumber *)a withB:(NSNumber *)b
{
   float number1 = [a floatValue];
   float number2 = [b floatValue];
   float product = number1 * number2;
   NSNumber *result = [NSNumber numberWithFloat:product];
   return result;
}

@end

int main()
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

   SampleClass *sampleClass = [[SampleClass alloc]init];
   NSNumber *a = [NSNumber numberWithFloat:10.5];
   NSNumber *b = [NSNumber numberWithFloat:10.0];
   NSNumber *result = [sampleClass multiplyA:a withB:b];
   NSString *resultString = [result stringValue];
   NSLog(@"The product is %@",resultString);

   [pool drain];
   return 0;
}
```

Exercice -5

créez une classe qui aura une méthode nommée « GénérateurNumerosRandom ». Cette fonction génère des nombres randome décimaux (entre 1 et 50) et calcule ensuite le totale, moyen et déviation standard. Convertissez ces 3 valeurs de la décimale en type integer puis retournez-les de cette fonction.
Maintenant, imprimez dans la commande ligne ces 3 valeurs (qui sont renvoyées par la fonction) dans la fonction « main ()»

Passer un tableaux dans un fonctions

Exercice -6

```
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject

/* function declaration */
-(double) getAverage:(int []) arr andSize:(int) size;

@end

@implementation SampleClass

-(double) getAverage:(int []) arr andSize:(int) size
{
  int    i;
  double avg;
  double sum =0;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = sum / size;

  return avg;
}

@end

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   /* pass pointer to the array as an argument */
   avg = [sampleClass getAverage:balance andSize: 5] ;

   /* output the returned value */
   NSLog( @"Average value is: %f ", avg );

   return 0;
}
```


Les diapositives sont faites à partir des matériaux à: www.tutorialpoint.com

Retourner un **tableaux** ou **pointer** de fonction

Exercice -7

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject

- (int *) getRandom;

@end

@implementation SampleClass

/* function to generate and return random numbers */
- (int *) getRandom
{
  static int  r[10];
  int i;

  /* set the seed */
  srand( (unsigned)time( NULL ) );
  for ( i = 0; i < 10; ++i)
  {
     r[i] = rand();
     NSLog( @"r[%d] = %d\n", i, r[i]);

  }

  return r;
}

@end

/* main function to call above defined function */
int main ()
{
   /* a pointer to an int */
   int *p;
   int i;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   p = [sampleClass getRandom];
   for ( i = 0; i < 10; i++ )
   {
       NSLog( @"*(p + %d) : %d\n", i, *(p + i));
   }

   return 0;
}
```

Pointeur vers un tableau

```objc
#import <Foundation/Foundation.h>

int main ()
{
   /* an array with 5 elements */
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   int i;

   p = balance;

   /* output each array element's value */
   NSLog( @"Array values using pointer\n");
   for ( i = 0; i < 5; i++ )
   {
       NSLog(@"*(p + %d) : %f\n",  i, *(p + i) );
   }

   NSLog(@"Array values using balance as address\n");
   for ( i = 0; i < 5; i++ )
   {
       NSLog(@"*(balance + %d) : %f\n",  i, *(balance + i) );
   }

   return 0;
}
```

Exercice -8

Envoyer un pointer dans un fonction en C

Exercice -9

Passer pointer dans un fonction

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
- (void) getSeconds:(int *)par;

@end
@implementation SampleClass

- (void) getSeconds:(int *)par{
 /* get the current number of seconds */
   *par = time( NULL );
   return;
}

@end

int main ()
{
   int sec;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass getSeconds:&sec];

   /* print the actual value */
   NSLog(@"Number of seconds: %d\n", sec );

   return 0;
}
```

Exercice -10

Passer pointer de tableaux dans un fonction

```objc
#import <Foundation/Foundation.h>

@interface SampleClass:NSObject
/* function declaration */
- (double) getAverage:(int *)arr ofSize:(int) size;
@end

@implementation SampleClass

- (double) getAverage:(int *)arr ofSize:(int) size
{
  int    i, sum = 0;
  double avg;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = (double)sum / size;

  return avg;
}

@end

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   /* pass pointer to the array as an argument */
   avg = [sampleClass getAverage: balance ofSize: 5 ] ;

   /* output the returned value  */
   NSLog(@"Average value is: %f\n", avg );

   return 0;
}
```

Structures en Objective-C

Exercice -11

Accéder le Structures

```
#import <Foundation/Foundation.h>

struct Books
{
   NSString *title;
   NSString *author;
   NSString *subject;
   int   book_id;
};

int main( )
{
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   Book1.title = @"Objective-C Programming";
   Book1.author = @"Nuha Ali";
   Book1.subject = @"Objective-C Programming Tutorial";
   Book1.book_id = 6495407;

   /* book 2 specification */
   Book2.title = @"Telecom Billing";
   Book2.author = @"Zara Ali";
   Book2.subject = @"Telecom Billing Tutorial";
   Book2.book_id = 6495700;

   /* print Book1 info */
   NSLog(@"Book 1 title : %@\n", Book1.title);
   NSLog(@"Book 1 author : %@\n", Book1.author);
   NSLog(@"Book 1 subject : %@\n", Book1.subject);
   NSLog(@"Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   NSLog(@"Book 2 title : %@\n", Book2.title);
   NSLog(@"Book 2 author : %@\n", Book2.author);
   NSLog(@"Book 2 subject : %@\n", Book2.subject);
   NSLog(@"Book 2 book_id : %d\n", Book2.book_id);

   return 0;
}
```

```
Book title : Objective-C Programming
Book author : Nuha Ali
Book subject : Objective-C Programming Tu
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Exercice -12

Structures comme les argument de fonctions

```
#import <Foundation/Foundation.h>

struct Books
{
   NSString *title;
   NSString *author;
   NSString *subject;
   int   book_id;
};

@interface SampleClass:NSObject

/* function declaration */
- (void) printBook:( struct Books) book ;

@end

@implementation SampleClass

- (void) printBook:( struct Books) book
{
   NSLog(@"Book title : %@\n", book.title);
   NSLog(@"Book author : %@\n", book.author);
   NSLog(@"Book subject : %@\n", book.subject);
   NSLog(@"Book book_id : %d\n", book.book_id);
}
@end

int main( )
{
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   Book1.title = @"Objective-C Programming";
   Book1.author = @"Nuha Ali";
   Book1.subject = @"Objective-C Programming Tutorial";
   Book1.book_id = 6495407;

   /* book 2 specification */
   Book2.title = @"Telecom Billing";
   Book2.author = @"Zara Ali";
   Book2.subject = @"Telecom Billing Tutorial";
   Book2.book_id = 6495700;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   /* print Book1 info */
   [sampleClass printBook: Book1];

   /* Print Book2 info */
   [sampleClass printBook: Book2];

   return 0;
}
```

Structures en Objective-C

Exercice -13

Pointers de Structure

```
#import <Foundation/Foundation.h>

struct Books
{
   NSString *title;
   NSString *author;
   NSString *subject;
   int   book_id;
};

@interface SampleClass:NSObject

/* function declaration */
- (void) printBook:( struct Books *) book ;

@end

@implementation SampleClass

- (void) printBook:( struct Books *) book
{
   NSLog(@"Book title : %@\n", book->title);
   NSLog(@"Book author : %@\n", book->author);
   NSLog(@"Book subject : %@\n", book->subject);
   NSLog(@"Book book_id : %d\n", book->book_id);
}
@end

int main( )
{
   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   Book1.title = @"Objective-C Programming";
   Book1.author = @"Nuha Ali";
   Book1.subject = @"Objective-C Programming Tutorial";
   Book1.book_id = 6495407;

   /* book 2 specification */
   Book2.title = @"Telecom Billing";
   Book2.author = @"Zara Ali";
   Book2.subject = @"Telecom Billing Tutorial";
   Book2.book_id = 6495700;

   SampleClass *sampleClass = [[SampleClass alloc]init];
   /* print Book1 info by passing address of Book1 */
   [sampleClass printBook:&Book1];

   /* print Book2 info by passing address of Book2 */
   [sampleClass printBook:&Book2];

   return 0;
}
```

```
Book title : Objective-C Programming
Book author : Nuha Ali
Book subject : Objective-C Programming T
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Les diapositives sont faites à partir des matériaux à:
www.tutorialpoint.com

Exercice -14

Tableaux

Créez une classe et cette classe doit avoir une fonction. Cette fonction aura un argument d'entrée de type chaîne de caractères (type **string**). Lorsque vous allez appeler cette fonction à partir de la fonction « main() », le paramètre d'entrée de cette fonction sera votre nom complet (prénom et nom de famille par exemple "Steve Jobs").

Cette fonction doit obtenir la valeur ASCII de tous les caractères de votre nom (rappelez d'ignorer l'espace entre votre prénom et votre nom de famille).

Maintiennent, convertir ces valeur en type « NSNumber»

Ce fonction doit retourner le total et la moyenne de ces valeurs ASCII de la fonction.

Imprimez maintenant ces 2 valeurs dans la fonction « main () ».

Peut-être cette fonction peut aider à obtenir la valeur ASCII de chaque caractère (il vaut la vérifier)

+ (NSNumber *)numberWithChar:(char)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un caractère signé.

Tableaux Multidimensionnelle

Exercice -15

Générer une matrice de 50 lignes et 20 colonnes de nombres aléatoires (random) dans la fonction « main () ». Maintenant dans la classe, créez 3 méthodes:

1ère méthode: Il devrait recevoir cette matrice 2D qui est créée dans la fonction « main () » (essayez d'envoyer cette matrice en utilisant le pointeur (peut-être que vous avez besoin d'un double pointeur). A l'intérieur de la fonction, calculer la moyenne de chaque ligne. Comme nous avons 50 lignes dans la matrice, nous obtiendrons 50 valeurs moyennes à partir de 50 lignes. Mettez ces valeurs dans un tableau et retournez-la de cette fonction (essayez à utiliser le pointer de tableaux pour retourner).
Imprimez maintenant ces valeurs dans la fonction "main () » comme ça :
« La moyenne de 1er ligne est : »
« La moyenne de 2em ligne est : »
…….
…….

Vous pouvez vous faire une idée de la création d'une matrice bidimensionnelle à partir de ce programme.

```
#import <Foundation/Foundation.h>

int main ()
{
   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ )
   {
      for ( j = 0; j < 2; j++ )
      {
         NSLog(@"a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }
   return 0;
}
```

Tableaux Multidimensionnelle

Exercice -15

2em méthode: Il devrait recevoir cette matrice 2D qui est créée dans la fonction « main () » (essayez d'envoyer cette matrice en utilisant le pointeur (peut-être que vous avez besoin d'un double pointeur). A l'intérieur de la fonction, calculer la moyenne de chaque colonne. Comme nous avons 20 colonne dans la matrice, nous obtiendrons 20 valeurs moyennes à partir de 20 colonne. Mettez ces valeurs dans un tableau et retournez-la de cette fonction (essayez à utiliser le pointer de tableaux pour retourner).

Imprimez maintenant ces valeurs dans la fonction "main () » comme ça :
« La moyenne de 1er colonne est : »
« La moyenne de 2em colonne est : »
……..
.....

3em méthode: Il devrait recevoir cette matrice 2D qui est créée dans la fonction « main () » (essayez d'envoyer cette matrice en utilisant le pointeur (peut-être que vous avez besoin d'un double pointeur). A l'intérieur de la fonction, calculer la moyenne de diagonale gauche et diagonale droite.

Mettez les valeurs de diagonale gauche dans un tableau et aussi mettez les valeurs de diagonale droite dans un tableau aussi. Retournez ces deux tableaux et deux valeur de moyenne de cette fonction (essayez à utiliser un structure qui contienne deux variable de type double et deux pointeur vers un tableaux. Utilisez aussi un pointeur du structure pour retourner de fonction).

Imprimez maintenant ces valeurs dans la fonction "main () » comme ça :
« La moyenne de diagonale gauche est: »
« La moyenne de diagonale droite est : »
« La 1er élément de diagonale gauche est: »
« La 2^{em} élément de diagonale gauche est: »
.....
.....
« La 1er élément de diagonale droite est: »
« La 2^{em} élément de diagonale droite est: »
.....
.....

Exercice -16

Section de String

écrire le programme complet avec toutes les fonctions mentionnées dans cette section "String". Les exemples sont présentés côte à côte.

Vous devez écrire des programmes complets en prenant l'aide des exemples présentés.

Créez des fonctions distinctes pour montrer l'application de chaque méthode présentée. Prenez les exemples / phrases / mots de votre choix pour montrer l'utilisation de ces méthodes

Strings en Objective-C

String représenter par NSString et NSMutableString

Méthode - 1

- (BOOL)isEqualToString:(NSString *)aString

Méthode - 2

(NSUInteger)length

Renvoie le nombre de caractères Unicode dans le récepteur

```
NSString *myString1=@"Hello World";
NSString *myString2=@"Hello World";

    if([myString1 isEqualToString:myString2]){

        NSLog(@"similar");
    }
    else{

        NSLog(@"dissimilar");
    }

     //  it returns a boolean, a special int
    NSLog(@"%i", [myString1 isEqualToString:myString2]);
```

```
NSString *welcome = @"Welcome!";

NSLog(@"%@", welcome);
```

```
NSString *uppercase = [@"welcome!" uppercaseString];

NSLog(@"%@", uppercase);
```

```
// the strings are NOT exactly alike

if ([@"snoop dog" isEqualToString:@"Snoop Dog"]) {
    NSLog(@"This will not print.");
} else {
    NSLog(@"Will the real Snoop Dog please stand up?");
}
```

```
/* total length of str3 after concatenation */
NSUInteger len = [str2 length];
NSLog(@"Length of Str2 :  %lu", (unsigned long)len );
```

```
NSUInteger welcomeLength = [@"welcome." length];

NSLog(@"welcomeLength: %lu", welcomeLength);
```

Strings en Objective-C

String représenter par NSString et NSMutableString

```
NSString* myString = @"10:Username taken";

if([myString hasPrefix:@"10"]) {
      //display more elegant error message
}
```

Méthode - 3

- **(BOOL)hasPrefix:(NSString *)aString**

Renvoie une valeur booléenne qui indique si une chaîne donnée correspond aux caractères de début du récepteur.

```
BOOL doctor = [@"Doctor Who" hasPrefix:@"Doctor"];

NSLog(@"doctor: %d", doctor);
```

Méthode - 4

- **(BOOL)hasSuffix:(NSString *)aString**

Renvoie une valeur booléenne qui indique si une chaîne donnée correspond aux caractères de fin du destinataire

```
BOOL esquire = [@"John Smith, Esq." hasSuffix:@"Esq."];

NSLog(@"esquire: %d", esquire);
```

Méthode - 5

- **(unichar)characterAtIndex:(NSUInteger)index**

Renvoie le caractère à une position de tableau donnée

```
unichar myVal = [otherString characterAtIndex:5];
NSLog(@"The char is :  %hu",myVal);
```

Méthode - 6

- **(id)initWithFormat:(NSString *)format ...**

Renvoie un objet NSString initialisé en utilisant une chaîne de format donnée en tant que modèle dans lequel les valeurs d'argument restantes sont remplacées

```
#import <Foundation/Foundation.h>

int main ()
{
   NSString *str1 = @"Hello";
   NSString *str2 = @"World";
   NSString *str3;
   int  len ;

   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

   /* InitWithFormat */
    str3 = [[NSString alloc] initWithFormat:@"%@ %@",str1,str2];
    NSLog(@"Using initWithFormat:   %@\n", str3 );
    [pool drain];

   return 0;
}
```

Strings en Objective-C

String représenter par NSString et NSMutableString

Méthode - 7

- **(NSInteger)integerValue**

Renvoie la valeur NSInteger du texte (de type NSNUmber, float, double etc.) du destinataire

Méthode - 8

- **(double)doubleValue**

Renvoie la valeur en virgule flottante du texte (de type NSNUmber, float, int etc.) du destinataire en double

Méthode - 9

- **(float)floatValue**

Renvoie la valeur en virgule flottante du texte (de type NSNUmber, double, int etc.) du destinataire en float

```
NSNumber *number = [NSNumber numberWithDouble:10.89555];
NSLog(@"%f",[number floatValue]); // To convert from NSNumber into float value
NSLog(@"%f",[number doubleValue]); // To convert from NSNulber into double value

NSNumber *numberInt = [NSNumber numberWithInteger:8955];
NSLog(@"%d",[numberInt intValue]); // To convert from NSNumber into integer value
```

```
NSString *ageString = @"29";
NSUInteger age = [ageString integerValue];

age++;  // birthday party!

NSLog(@"%lu", age);
```

Strings en Objective-C

String représenter par NSString et NSMutableString

Méthode - 10

- **(NSString *)uppercaseString**

  Retourne une représentation en majuscule du récepteur

Méthode - 11

- **(NSString *)lowercaseString**

  Retourne une représentation en minuscule du récepteur

Méthode - 12

- **(NSString *)capitalizedString**

  Renvoie une représentation en majuscule du destinataire

```
NSString *welcome = [@"welcome." uppercaseString];

NSLog(@"%@", welcome);
```

```
NSString *welcome = [@"WELCOME!" lowercaseString];

NSLog(@"%@", welcome);
```

```
27 int main(int argc, const char * argv[]) {
28     @autoreleasepool {
29
30         NSString *str1 = @"Bonjour";
31         NSString *str2 = @"Salut ça va";
32
33         NSString *str3;
34         NSString *str4, *str5;
35
36         /* Converting into Upper Case String */
37         str3 = [str2 uppercaseString];
38         NSLog(@"The Uppercase String is :  %@",str3);
39
40         /* Converting into Lower Case String */
41         str4 = [str1 lowercaseString];
42         NSLog(@"The Lowercase String is :  %@",str4);
43
44
45         /* Converting into Upper Case String */
46         str5 = [str2 capitalizedString];
47         NSLog(@"The Capitalized String is :  %@",str3);
48
```

```
NSString *welcomeToFlatiron = [@"welcome to flatiron." capitalizedString];

NSLog(@"%@", welcomeToFlatiron);
```

Strings en Objective-C

Méthode - 13

- **(NSRange)rangeOfString:(NSString *)aString**

Trouve et renvoie la plage de la première occurrence d'une chaîne donnée dans le récepteur.

```
NSString *string = @"hello bla bla";
if ([string rangeOfString:@"bla"].location == NSNotFound) {
  NSLog(@"string does not contain bla");
} else {
  NSLog(@"string contains bla!");
}
```

```
NSRange range = [name rangeOfString: @"Arnold"];
NSUInteger start = range.location;
NSUInteger end = start + range.length;
```

Méthode - 14

- **(NSString *)stringByAppendingFormat:(NSString *)format**

Renvoie une chaîne créée en ajoutant au récepteur une chaîne construite à partir d'une chaîne de format donnée et les arguments suivants

```
NSString *aString =self.stringName;
NSString *resultString =[name stringByAppendingFormat:@"%@",aString];
```

```
NSString *myString = @"some text:  ";
myString = [myString stringByAppendingFormat:@" à quelle heure, Tu vas aller pour badminton = %d", 3];
NSLog(@"%@",myString);
```

Strings en Objective-C

Méthode - 15

- **(NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set**

Renvoie une nouvelle chaîne créée en supprimant des deux extrémités du récepteur les caractères contenus dans un jeu de caractères donné

```
NSString *string = @" this text has spaces before and after ";
NSString *trimmedString = [string stringByTrimmingCharactersInSet:
                                  [NSCharacterSet whitespaceCharacterSet]];
```

Méthode - 16

- **(NSString *)substringFromIndex:(NSUInteger)anIndex**

Renvoie une nouvelle chaîne contenant les caractères du destinataire de celui d'un index donné à la fin.

```
NSString *myString = @"This is my String";
myString = [myString substringFromIndex:5];
```

Fin de la section de String

typedef vs #define

Exercice -17

Ecrivez un programme pour montrer l'application et la différence entre **#define** et **typedef**.

Nous devrions pouvoir comprendre la différence entre ces 2 mots clés de votre code/programme.

```
#import <Foundation/Foundation.h>

#define TRUE  1
#define FALSE 0

int main( )
{
   NSLog( @"Value of TRUE : %d\n", TRUE);
   NSLog( @"Value of FALSE : %d\n", FALSE);

   return 0;
}
```

```
#import <Foundation/Foundation.h>

typedef struct Books
{
   NSString *title;
   NSString *author;
   NSString *subject;
   int book_id;

} Book;

int main( )
{

   Book book;
   book.title = @"Objective-C Programming";
   book.author = @"TutorialsPoint";
   book.subject = @"Programming tutorial";
   book.book_id = 100;
   NSLog( @"Book title : %@\n", book.title);
   NSLog( @"Book author : %@\n", book.author);
   NSLog( @"Book subject : %@\n", book.subject);
   NSLog( @"Book Id : %d\n", book.book_id);

   return 0;
}
```

```
Book title : Objective-C Programming
Book author : TutorialsPoint
Book subject : Programming tutorial
Book Id : 100
```

Heritage

```objc
#import <Foundation/Foundation.h>

@interface Person : NSObject

{
    NSString *personName;
    NSInteger personAge;
}

- (id)initWithName:(NSString *)name andAge:(NSInteger)age;
- (void)print;
@end

@implementation Person

- (id)initWithName:(NSString *)name andAge:(NSInteger)age{
    personName = name;
    personAge = age;
    return self;
}

- (void)print{
    NSLog(@"Name: %@", personName);
    NSLog(@"Age: %ld", personAge);
}

@end

@interface Employee : Person

{
    NSString *employeeEducation;
}

- (id)initWithName:(NSString *)name andAge:(NSInteger)age
  andEducation:(NSString *)education;
- (void)print;

@end


@implementation Employee

- (id)initWithName:(NSString *)name andAge:(NSInteger)age
  andEducation: (NSString *)education
  {
    personName = name;
    personAge = age;
    employeeEducation = education;
    return self;
}

- (void)print
{
    NSLog(@"Name: %@", personName);
    NSLog(@"Age: %ld", personAge);
    NSLog(@"Education: %@", employeeEducation);
}

@end


int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Base class Person Object");
    Person *person = [[Person alloc]initWithName:@"Raj" andAge:5];
    [person print];
    NSLog(@"Inherited Class Employee Object");
    Employee *employee = [[Employee alloc]initWithName:@"Raj"
    andAge:5 andEducation:@"MBA"];
    [employee print];
    [pool drain];
    return 0;
}
```

Exercice -18

Créez 3 classes. Avoir une fonction (fonc-1) dans la classe 1, mettre 2-3 plus de fonction dans la classe 1.

Maintenant, la classe 2 hériterait de la classe 1 et devrait aussi avoir une fonction appelée (func-1) avec d'autres fonctions (2-3).

Par la suite, dans la classe 3 aussi, nous devrions avoir quelques fonctions (2-3) et puis aussi avoir func-1.
C'est ce qu'on appelle la fonction prioritaire.

Montrer maintenant dans la fonction main () la fonction du processus pour accéder au func-1 de chaque classe individuellement

vous pouvez recevoir de l'aide d'ici ou de la diapositive suivante

```
Base class Person Object
Name: Raj
Age: 5
Inherited Class Employee Object
Name: Raj
Age: 5
Education: MBA
```

Cherche sur internet pour l'aide

Exercice -19

Polymorphisme

```
#import <Foundation/Foundation.h>

@interface Shape : NSObject

{
    CGFloat area;
}

- (void)printArea;
- (void)calculateArea;
@end

@implementation Shape

- (void)printArea{
    NSLog(@"The area is %f", area);
}

- (void)calculateArea{

}

@end


@interface Square : Shape
{
    CGFloat length;
}

- (id)initWithSide:(CGFloat)side;

- (void)calculateArea;

@end
```

```
@implementation Square

- (id)initWithSide:(CGFloat)side{
    length = side;
    return self;
}

- (void)calculateArea{
    area = length * length;
}

- (void)printArea{
    NSLog(@"The area of square is %f", area);
}

@end

@interface Rectangle : Shape
{
    CGFloat length;
    CGFloat breadth;
}

- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth;


@end
```

```
@implementation Rectangle

- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth{
    length = rLength;
    breadth = rBreadth;
    return self;
}

- (void)calculateArea{
    area = length * breadth;
}

@end


int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Shape *square = [[Square alloc]initWithSide:10.0];
    [square calculateArea];
    [square printArea];
    Shape *rect = [[Rectangle alloc]
    initWithLength:10.0 andBreadth:5.0];
    [rect calculateArea];
    [rect printArea];
    [pool drain];
    return 0;
}
```

Les diapositives sont faites à partir des matériaux à: www.tutorialpoint.com

Exercice -20

Catégories

```
#import <Foundation/Foundation.h>

@interface NSString(MyAdditions)

+(NSString *)getCopyRightString;

@end

@implementation NSString(MyAdditions)

+(NSString *)getCopyRightString{
    return @"Copyright TutorialsPoint.com 2013";
}

@end


int main(int argc, const char * argv[])
{

   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   NSString *copyrightString = [NSString getCopyRightString];
   NSLog(@"Accessing Category: %@",copyrightString);
   [pool drain];
    return 0;
}
```

Exercice -20

Un peu plus des exemples du Catégories

La méthode **reverseString** ajoute la possibilité à tous les objets **NSString** d'inverser les caractères de la chaîne.

```
@interface NSString (reverse)
-(NSString *) reverseString;
@end
```

Comme pour la déclaration **@interface**, la section **@implementation** ne change que dans la mesure où le nom de la catégorie est ajouté.

```
@implementation NSString (reverse)

-(NSString *) reverseString
{
  NSMutableString *reversedStr;
  int len = [self length];

  // Auto released string
  reversedStr = [NSMutableString stringWithCapacity:len];

  // Probably woefully inefficient...
  while (len > 0)
    [reversedStr appendString:
         [NSString stringWithFormat:@"%C", [self characterAtIndex:--len]]];

  return reversedStr;
}

@end
```

Comment utiliser la catégorie

```
#import <Foundation/Foundation.h>
#import "NSString+Reverse.h"

int main (int argc, const char * argv[])
{
  NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
  NSString *str  = [NSString stringWithString:@"Fubar"];
  NSString *rev;

  NSLog(@"String: %@", str);
  rev = [str reverseString];
  NSLog(@"Reversed: %@",rev);

  [pool drain];
  return 0;
}
```

convention de nommage

- NSString+Reverse.h
- NSString+Reverse.m

Extensions

Exercice -20

```
#import <Foundation/Foundation.h>

@interface SampleClass : NSObject
{
    NSString *name;
}

- (void)setInternalID;
- (NSString *)getExternalID;

@end


@interface SampleClass()
{
    NSString *internalID;
}

@end

@implementation SampleClass

- (void)setInternalID{
    internalID = [NSString stringWithFormat:
    @"UNIQUEINTERNALKEY%dUNIQUEINTERNALKEY",arc4random()%100];
}

- (NSString *)getExternalID{
    return [internalID stringByReplacingOccurrencesOfString:
    @"UNIQUEINTERNALKEY" withString:@""];
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    [sampleClass setInternalID];
    NSLog(@"ExternalID: %@",[sampleClass getExternalID]);
    [pool drain];
    return 0;
}
```

Exercice -21

Ecrire 2 programmes pour montrer l'utilisation des catégories

Exercice -22

Ecrire 2 programmes pour montrer l'utilisation des protocoles

Protocoles

Exercice -23

```objc
#import <Foundation/Foundation.h>

@protocol PrintProtocolDelegate

- (void)processCompleted;

@end

@interface PrintClass :NSObject
{
    id delegate;
}

- (void) printDetails;
- (void) setDelegate:(id)newDelegate;
@end

@implementation PrintClass

- (void)printDetails{
    NSLog(@"Printing Details");
    [delegate processCompleted];
}

- (void) setDelegate:(id)newDelegate{
    delegate = newDelegate;
}

@end
```

```objc
@interface SampleClass:NSObject<PrintProtocolDelegate>

- (void)startAction;

@end

@implementation SampleClass

- (void)startAction{
    PrintClass *printClass = [[PrintClass alloc]init];
    [printClass setDelegate:self];
    [printClass printDetails];
}

-(void)processCompleted{
    NSLog(@"Printing Process Completed");
}

@end


int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    [sampleClass startAction];
    [pool drain];
    return 0;
}
```

Section de Préprocesseur

Exercice -24

Préprocesseur

Ecrivez le programme de votre choix pour montrer l'utilisation de ces mots clés mentionnés dans le tableau suivant. Dans les autres diapositives de cette section, vous trouverez l'exemple d'utilisation de ces mots clés. Vous pouvez recevoir de l'aide de leur part mais n'écrivez pas la même ligne de code que dans l'exemple. Vous devez trouver vos propres exemples (prenez l'aide d'Internet) et être en mesure de montrer clairement l'utilisation de ces mots clés dans votre programme.

#define	Substitute une macro de préprocesseur
#include	Insère un en-tête particulier d'un autre fichier
#undef	Undefines une macro de préprocesseur
#ifdef	Renvoie true si cette macro est définie
#ifndef	Reenvoie true si cette macro n'est pas définie
#if	Teste si une condition est vraie ou pas pendant compilation
#else	L'alternative pour #if
#eif	#else un #if dans une déclaration
#endif	Termine le préprocesseur conditionnel
#error	Imprime un message d'erreur sur stderr
#pragma	Emet des commandes spéciales au compilateur en utilisant une méthode standardisée

Préprocesseur

-- remplacer les instances de MAX_ARRAY_LENGTH par 20

```
#define MAX_ARRAY_LENGTH 20
```

-- Obtenir foundation. h de Foundation Framework
-- Récupérer myheader. h du répertoire local et d'ajouter le contenu

```
#import <Foundation/Foundation.h>
#include "myheader.h"
```

-- Cela indique à ne pas définir FILE_SIZE existant et de le définir comme 42

```
#undef  FILE_SIZE
#define FILE_SIZE 42
```

-- Ceci indique à ne définir MESSAGE que si MESSAGE n'est pas déjà défini.

```
#ifndef MESSAGE
   #define MESSAGE "You wish!"
#endif
```

-- Ceci indique à faire le processus les instructions jointes si DEBUG est défini

```
#ifdef DEBUG
   /* Your debugging statements here */
#endif
```

Macros prédéfinies

```
#import <Foundation/Foundation.h>

int main()
{
   NSLog(@"File :%s\n", __FILE__ );
   NSLog(@"Date :%s\n", __DATE__ );
   NSLog(@"Time :%s\n", __TIME__ );
   NSLog(@"Line :%d\n", __LINE__ );
   NSLog(@"ANSI :%d\n", __STDC__ );

   return 0;
}
```

Opérateurs de préprocesseurs

Macro Continuation : L'opérateur de macro continuation est utilisé pour continuer une macro trop longue pour une seule ligne.

```
#define  message_for(a, b)  \
    NSLog(@#a " and " #b ": We love you!\n")
```

```
#define NINE (3 \
  + 3)
```

→ It's Six

```
#include <stdio.h>

#define COMPARE(x,y)   ((x)<(y)?-1:((x)==(y)?0:1))

int main()
{
  double a = 1.5;
  int n = 2;
  float z = 1.1f;
  printf("\n compare %lf (double) with %f (float): %d", a , z, COMPARE(a,z));
  printf("\n compare %lf (double) with %d (int): %d", a , n, COMPARE(a,n));
  printf("\n compare %d (int) with %f (float): %d", n , z, COMPARE(n,z));
  printf("\n compare \" n * a\" = %lf \n"
    " with \"n+1\" = to %d (double): %d",
     n*a , n+1, COMPARE(n*a,n+1));
  printf("\n");
}
```

Opérateurs de préprocesseurs

Stringize/Signe Numérique (#):

```
#import <Foundation/Foundation.h>

#define  message_for(a, b)  \
    NSLog(@#a " and " #b ": We love you!\n")

int main(void)
{
   message_for(Carole, Debra);
   return 0;
}
```

```
2013-09-14 05:46:14.859 demo[20683] Carole and Debra: We love you!
```

Opérateurs de préprocesseurs

L'opérateur de collage de jetons (##):

```
#import <Foundation/Foundation.h>

#define tokenpaster(n) NSLog (@"token" #n " = %d", token##n)

int main(void)
{
   int token34 = 40;

   tokenpaster(34);
   return 0;
}
```

```
2013-09-14 05:46:14.859 demo[20683] Carole and Debra: We love you!
```

Opérateurs de préprocesseurs

L'opérateur defined ():

```
#import <Foundation/Foundation.h>

#if !defined (MESSAGE)
   #define MESSAGE "You wish!"
#endif

int main(void)
{
   NSLog(@"Here is the message: %s\n", MESSAGE);
   return 0;
}
```

```
2013-09-14 05:48:19.859 demo[20683] Here is the message: You wish!
```

Opérateurs de préprocesseurs

```
#import <Foundation/Foundation.h>

#define MAX(x,y) ((x) > (y) ? (x) : (y))

int main(void)
{
   NSLog(@"Max between 20 and 10 is %d\n", MAX(10, 20));
   return 0;
}
```

```
2013-09-14 05:52:15.859 demo[20683] Max between 20 and 10 is 20
```

Fin de section de préprocesseur

Type casting

Promotion de Integer

Exercice -25

-- Le division de deux valeur Integer

```
#import <Foundation/Foundation.h>

int main()
{
   int sum = 17, count = 5;
   CGFloat mean;

   mean = (CGFloat) sum / count;
   NSLog(@"Value of mean : %f\n", mean );

   return 0;
}
```

```
Value of mean : 3.400000
```

Exercice -26

```
#import <Foundation/Foundation.h>

int main()
{
   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   int sum;

   sum = i + c;
   NSLog(@"Value of sum : %d\n", sum );

   return 0;
}
```

```
Value of sum : 116
```

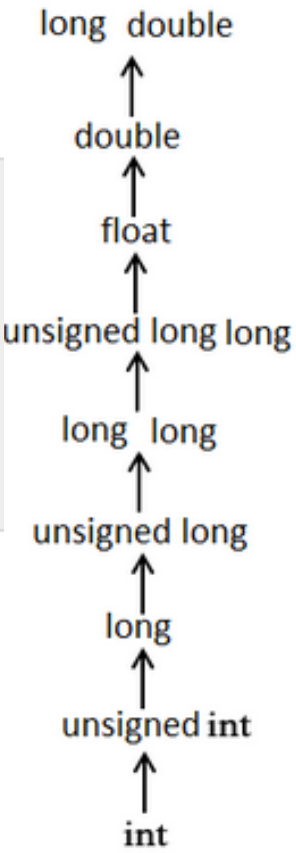
Exercice -27

```
#import <Foundation/Foundation.h>

int main()
{
   int  i = 17;
   char c = 'c'; /* ascii value is 99 */
   CGFloat sum;

   sum = i + c;
   NSLog(@"Value of sum : %f\n", sum );
   return 0;
}
```

```
Value of sum : 116.000000
```



essayer de comprendre la différence entre les programmes 25, 26 et 27