Nom / Prénom : _____

1. Déclarez une variable de chaque type dont vous vous souvenez et initialisez les à une valeur correspondant au type utilisé.

Rappel de cours :

Pour déclarer une variable, on utilise la syntaxe suivante :

```
typeVariable nomVariable;
```

On remplace *typeVariable* par un des types connus : **int, long, float, double, boolean, char, String**
Il est également possible d'initialiser directement la variable par exemple :

```
String resolution = "Ne jamais remettre au lendemain ...";
```

Il est possible de déclarer plusieurs variables ***du même type*** sur une seule ligne:

```
typeVariable nomVar1, nomVar2, nomVar3;
```

Dans ce cas il est possible d'initialiser (ou non) les variables indépendamment en écrivant :

```
int a = 5, b, c = 0, d, e = 423, f, g;
```

Une fois déclarée, il est possible de modifier la valeur d'une variable en utilisant l'*affectation.* Celle-ci **écrase** alors le contenu précédent de la variable.
L'affectation peut contenir une simple valeur ou des expressions plus complexes :

```
int variAff = 5;
variAff = 1664;
variAff = (autreVar * 2) / 23;
```

Enfin il est possible de stocker dans une variable le résultat d'une fonction **si le type de retour de cette fonction correspond au type de la variable**.

```
double uneVar = 1.664;
uneVar = calculeUnTruc(23.05, 42);
uneVar = calculeTruc(2.64) * autreVar;
```

int ageEtudiant = 20;
long vitesseLumiere = 299792458;
float moyenneAges = 21.234;
double pi = 3.14159262637;
boolean pile = true;
char sigleDollars = '$';
String retenirPi = "Que j'aime à faire connaître ce nombre si utile aux sages";

2. Déclarez une fonction *moyenneAires()* qui prend en argument 2 rayons et calcule la moyenne de l'aire des 2 cercles correspondants. On rappelle que la formule de calcul de l'aire est $a = \pi R^2$
La valeur de π est disponible en Java en utilisant *Maths.PI* (type **double**)

Rappel de cours :

Pour déclarer une fonction, on utilise la syntaxe suivante :

```
public static typeRetour nomFonction(typeArg1 arg1, typeArg2 arg2, ...) { ... }
```

Encore une fois, on remplace *typeRetour, typeArg1*, ... par un des types connus.
Si on remplace le *typeRetour* par **void**, cela signifie que **la fonction ne renverra rien.** (pas d'instruction **return**)

```
public static void fonctionInutile()
{
    System.out.println("Je ne sers à rien");
}
```

A l'inverse lorsqu'on remplace *typeRetour* par n'importe quel autre type, la fonction devra renvoyer une valeur correspondant au type défini. **Attention** dans ce cas **l'instruction *return* est obligatoire.**

```
public static double multiplieMoiPar4(double moi)
{
    double resultat = moi * 4;
    return resultat;
}
```

Attention encore, l'instruction *return* signifie la fin de la fonction :

```
public static int renvoie164()
{
    return 164;
    System.out.println("Je serais jamais affiché");
}
```

Une fois la fonction définie, il est alors possible de l'*appeller* ce qui permettra de réaliser les instructions contenues dans celle-ci. Il suffit dans ce cas de donner des arguments à la fonction qui sont du bon type :

```
double variMoi;
variMoi = multiplieMoiPar4(16);
variMoi = multiplieMoiPar4(variMoi);
```

```
public static double moyenneAires(double rayon1, double rayon2)
{
    double aire1 = Maths.PI * rayon1 * rayon1;
    double aire2 = Maths.PI * rayon2 * rayon2;
    return (aire1 + aire2) / 2;
}
```

3. Déclarez une fonction *romainToInt()* qui prend en argument une chaîne de caractères pouvant contenir les chiffres romains *"I" "II" "III"* ou *"IV"* et renvoie la valeur entière correspondante.
On rappelle que pour tester qu'une chaîne de caractères *nom* est égale à *"Florence"* on utilise la fonction *nom.equals("Florence")*

Rappel de cours :

L'utilisation de l'alternative permet de modifier l'exécution des instructions en fonction des cas rencontrés.
On utilise une **expression de test booléenne** *e* qui peut être vraie ou fausse

```
if (e)
  I1;
else
  I2;
```

```
if (i == 1)
  s = "i est égal à 1";
else
  s = "i est différent de 1";
```

L'instruction else est facultative, il est donc possible d'écrire :

```
if (e)
  I1;
```

```
if (i == 1)
  s = "i est égal à 1";
```

Pour réaliser plusieurs instructions dans une alternative, il **faut les entourer par { et }**

```
if (e)
{
  I1;
   ...
}
```

```
if (i == 1)
{
  s = "i est égal à 1";
  i = j;
}
```

```
if (i==1)
{
    s=" i est égal à 1 ";
    i=j;
}
else
{
    s=" i est différent de 1 ";
    i=1515;
}
```

Enfin il est possible de cascader les conditions et de combiner les alternatives de toutes les manières imaginables

```
if (i==1)
    s = "i est égal à 1";
else if (i==2)
    s = "i est égal à 2";
else if (i==3)
    s = "i est égal à 3";
else
    s = "i est différent de 1,2 et 3";
```

```
public static int romainToInt(String chiffreRomain)
{
   if (chiffreRomain.equals("I"))
      return 1;
   if (chiffreRomain.equals("II"))
      return 2;
   if (chiffreRomain.equals("III"))
      return 3;
   if (chiffreRomain.equals("IV"))
      return 4;
   return 0;
}
```

4. Appellez la fonction précédente pour effectuer l'opération *I + IV* et afficher le résultat

```
int resultat = arabifie("I") + arabifie("IV");
System.out.println(resultat);
```

5. Ecrire le test booléen permettant de réaliser le *OU exclusif* entre deux variables booléennes **a** et **b**.

Rappel de cours :

Les tests booléens permettent d'utiliser la logique booléenne et sont indispensables pour les structures de contrôle comme l'alternative. Ceux-ci ne peuvent donc renvoyer que les valeurs vrai (**true)** ou faux (**false**). On distingue deux types d'opérateurs pour les tests booléens**.**

Opérateurs relationnels
Ils permettent tester tous les types **avec une relation d'ordre** : entiers, flottants, caractères, ... et renvoient une valeur booléenne

opérateurs relationnels	action	exemple
<	plus petit que	x < i;
>	plus grand que	i > 100;
<=	plus petit ou égal à	j <= k;
>=	plus grand ou égal à	c >= 'a';
==	égal à	i == 20;
!=	différent de	c != 'z';

Attention à bien utiliser l'opérateur == pour tester l'égalité

Opérateurs logiques

Les opérateurs logiques permettent de "connecter" des opérandes booléennes. On a donc les trois opérateurs de base de la logique booléenne:

- ! la négation
- && le ET
- || le OU

p	q	!p	p && q	p || q
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

On peut alors combiner toutes les opérandes booléennes possible :

```
varBool && !(i > 10 || (i < 5 && i > x))
```

A noter qu'il est évidemment possibler d'utiliser une fonction qui renvoie un type **boolean** dans un test

```
renvoieTrucVraiOuFaux() && x == 23
```

L'évaluation de l'expression logique est stoppée dès que sa valeur de vérité peut être assurée.
Dans le cas `p1 && p2 && ... & pn`
l'évaluation est stoppée si `pi` est évaluée à faux.
Dans le cas `p1 || p2 || ... || pn`
l'évaluation est stoppée si `pi` est évaluée à vrai.

```
(a && !b) || (!a && b)
```