Informatique *TD*
Programmation fonctionnelle avec OCaml.1 Ipesup

Objectifs

- Maîtriser les déclarations simples en OCaml et comprendre la portée lexicale.
- Expérimenter l'inférence de types et identifier les erreurs de typage.
- Manipuler des expressions simples (int, float, bool, string, char).
- Utiliser les conditions et écrire de petites fonctions.
- Pratiquer l'affichage, la lecture, et les séquences d'instructions.
- Découvrir les modules de base (`String`, `List`, `Printf`).

Exercice

Écrire et prédire le type de chaque déclaration :

```
let a = 42
let b = -17
let c = 3.14
let d = -2.5e-3
let e = true
let f = false
let g = 'A'
let h = "Bonjour"
```

Vérifier avec le compilateur.

Exercice

Expliquer pourquoi la portée est dite *lexicale* et comparer :

```
let x = 42
let y = x * x + 10
```

et

```
let x = 42
let x = 100
```

Exercice

Évaluer et donner le résultat attendu :

1. `5 + 3`, `5 - 8`, `7 * 6`, `17 / 5`, `17 mod 5`.
2. `2.5 +. 3.0`, `7. /. 2.`, `(2.0 ** 3.) +. 1.`.
3. `not true`, `true && false`, `(3 < 5) || (10 = 2)`.
4. `"Hello " ^"World"`, `;"abc".[0]`, `String.length "test"`.

Exercice

Tester (et corriger) les erreurs de typage :

```
let x = 5 + 2.0
let y = "abc" + "def"
```

Exercice

Afficher les valeurs suivantes :

1. `print_int 42`, puis sauter une ligne avec `print_newline ()`.
2. `Printf.printf "x = %d et y = %f \n" 42 3.14`.
3. `print_char 'Z'`, `print_string "Bonjour"`.

Exercice

Écrire une fonction `carre : int -> int`.
Tester avec 2, 5, -3. Comparer `carre 2 + 1` et `carre (2 + 1)`.

Exercice

Écrire un programme qui lit un entier et affiche :

- `"négatif"` si le nombre est < 0,
- `"zéro"` si le nombre est $= 0$,
- `"positif pair"` si le nombre est > 0 et pair,
- `"positif impair"` si le nombre est > 0 et impair.

Exercice

Définir une fonction `hello : unit -> unit` qui affiche `"Hello World"`. Pourquoi son type est-il `unit -> unit` ?

Exercice

Définir une fonction `date : unit -> unit` qui affiche la date d'aujourd'hui.

Exercice

Écrire un programme interactif :

1. Lire un nom et un date de naissance avec.
2. Afficher `"Bonjour <nom>, nous sommes le ... et tu as ... ans"`.
3. Demander `"As-tu un événement aujourd'hui ? (oui/non)"` et lire la réponse.
4. Afficher une phrase différente selon la réponse (oui -> `"Bon courage"`, non -> `"Relax!"`)

Exercice

Écrire :

```
let res =
  let x = 10 in
  let y = x + 5 in
  y * 2
```

Quelle valeur obtient-on ?

Exercice

Tester l'usage de `;` :

```
print_string "Bonjour"; print_newline ();
print_string "Au revoir"; print_newline ()
```

Expliquer pourquoi le résultat est `unit`.

Exercice

Écrire un programme qui affiche trois lignes de texte différentes (`"Bonjour"`, `"Comment ça va ?"`, `"Au revoir"`) en utilisant plusieurs appels à `print_string` séparés par `;`.

Exercice

Définir une fonction `saluer : unit -> unit` qui exécute en séquence :

1. Affiche `"=== Début du programme ==="`.
2. Demande le nom de l'utilisateur.
3. Affiche `"Ordi de <nom>"`.
4. Affiche `"=== Fin du programme ==="`.

Utiliser un bloc `begin ... end` pour grouper plusieurs instructions.

Exercice

Écrire une fonction `afficher_3fois : string -> unit` qui affiche trois fois la même chaîne donnée en argument, en une seule définition avec des séquences d'instructions.

Exercice

Écrire un petit programme qui lit un entier avec `read_int`, puis exécute en séquence :

1. Affiche `"Tu as entré <n>"`.
2. Affiche le double de `n`.
3. Affiche le carré de `n`.

Exercice

Créer une fonction `menu : unit -> unit` qui affiche successivement des lignes de menu (`"1. Calculer"`, `"2. Quitter"`) et lit le choix avec `read_int`.

Exercice

Avec le module `String` :

1. Lire une chaîne avec `read_line`.
2. Afficher sa longueur avec `String.length`.
3. Afficher son premier caractère `s.[0]`.

Exercice

Avec le module `List` (sans entrer dans les détails des listes) :
Définir `let l = [1;2;3;4;5]`. Tester `List.length l`.

Exercice

Écrire une fonction `bilan : float -> unit` qui lit une moyenne (sur 20) et affiche :

— `"Échec"` si < 10,
— `"Passable"` si entre 10 et 12,
— `"Assez bien"` si entre 12 et 14,
— `"Bien"` si entre 14 et 16,
— `"Très bien"` si ≥ 16.

Exercice

Écrire un programme interactif :

1. Lit un nom avec `read_line`.
2. Lit un âge avec `read_int`.
3. Si l'âge < 18, affiche `"<nom>, tu es mineur"`.
4. Sinon, affiche `"<nom>, tu es majeur"`.

Exercice

Créer une fonction `calculatrice : unit -> unit` qui :

1. Demande deux entiers a et b.
2. Demande à l'utilisateur de choisir une opération : somme, différence, produit ou quotient.
3. Utilise un `if...then...else` imbriqué pour exécuter l'opération choisie.
4. Affiche le résultat avec `Printf.printf`.