SYNTAX FOR TYPE ABBREVIATIONS (4/4 points)

Warning: you only have 2 attempts. Some hints will be displayed after the first attempt.

How will the OCaml compiler react to the following definition: `type A = int;;` ?

- (●) It will report a *syntax error*. ✔
- ( ) It will report a *type error*.
- ( ) It will *accept* the definition.

A type identifier must start with a lowercase letter.

How will the OCaml compiler react to the following definition: `type t = int;;` ?

- ( ) It will report a *syntax error*.
- ( ) It will report a *type error*.
- (●) It will *accept* the definition. ✔

There is no syntax error here, `int` is a valid type expression.

How will the OCaml compiler react to the following definition: `type a = boolean;;` ?

- ( ) It will report a *syntax error*.
- (●) It will report a *type error*. ✔

The type for booleans is written `bool`. In the initial typing environment, `boolean` is not a valid type expression.

How will the OCaml compiler react to the following definition: `type a, b = int;;`?

- (●) It will report a *syntax error*. ✔
- ( ) It will report a *type error*.
- ( ) It will *accept* the definition.

Only one type identifier can be introduced in a type abbreviation.

Vous avez utilisé 2 essais sur 2

SYNTAX FOR TYPE ANNOTATIONS (3/3 points)

Warning: you only have 2 attempts. Some hints will be displayed after the first attempt.

How will the OCaml compiler react to the following definition: `let f (int x) = x;;`?

- (●) It will report a *syntax error*. ✔
- ( ) It will report a *type error*.
- ( ) It will *accept* the definition.

To assign the type `int` to the argument `x`, one must write `(x : int)` (the parenthesis are mandatory). A valid rewriting would be `let f (x : int) = x;;`.

`let f (x : int) :: int = x;;`?

- [x] It will report a *syntax error*. ✔
- [ ] It will report a *type error*.
- [ ] It will *accept* the definition.

Only one colon is required to describe the return type of a function. A valid definition for a function of type `int -> int` would be: `let f (x : int) : int = x;;`.

How will the OCaml compiler react to the following definition: `let f x = (x : bool);;`?

- [ ] It will report a *syntax error*.
- [ ] It will report a *type error*.
- [x] It will *accept* the definition. ✔

There is no syntax error here, any expression might annotated; this a function of type `bool -> bool`.

Vous avez utilisé 1 essais sur 2

SYNTAX FOR TYPE ANNOTATIONS (BIS) (4/4 points)

Warning: you only have 2 attempts. Some hints will be displayed after the first attempt.

How will the OCaml compiler react to the following definition: `let f (x : bool) : bool = x;;`?

- It will report a *type error*.
- It will *accept* the definition. ✔

There is no syntax error here; this a function of type `bool -> bool`.

How will the OCaml compiler react to the following definition: `let f (x : int) : float = x;;`?

- It will report a *syntax error*.
- It will report a *type error*. ✔
- It will *accept* the definition.

There is no syntax error here; but we try to type the variable `x` once with `int` (as an argument) and once with `float` (as the returned value).

How will the OCaml compiler react to the following definition: `let f x y = if x then (y : int) else y + 1;;`?

- It will report a *syntax error*.
- It will report a *type error*.
- It will *accept* the definition. ✔

- There is no syntax error here, any expression might be annotated; this a function of type `bool -> int -> int`.

How will the OCaml compiler react to the following definition:
`let f (x : string) : char = String.get x 0;;` ?

- It will report a *syntax error*.
- It will report a *type error*.
- It will *accept* the definition. ✔

There is no syntax error here; this is a function of type:
`string -> char`.

Vous avez utilisé 1 essais sur 2

TO SUM UP (6/6 points)

How will the OCaml compiler react to the following definition:
`type b = bool;;` ?

- It will report a *syntax error*.
- It will report a *type error*.
- It will *accept* the definition. ✔

There is no syntax error here.

How will the OCaml compiler react to the following definition:
`type F = float;;` ?

- It will report a *syntax error*. ✔

- It will *accept* the definition.

A type identifier must start with a lowercase letter.

How will the OCaml compiler react to the following definition: `let positive x : bool = (x > 0);;`?

- It will report a *syntax error*.
- It will report a *type error*.
- It will *accept* the definition. ✔

There is no syntax error here. In this position, the `bool` annotation describes the return type of the function, not the argument `x`.

How will the OCaml compiler react to the following definition: `let f (bool x) = x;;`?

- It will report a *syntax error*. ✔
- It will report a *type error*.
- It will *accept* the definition.

To assign the type `bool` to the argument `x`, the correct syntax is `let f (x : bool) = x;;`.

How will the OCaml compiler react to the following definition: `let f (x : int) : bool = x;;`?

- [x] It will report a *type error*. ✔
- [ ] It will *accept* the definition.

There is no syntax error here, just two incompatible type annotations about `x`.

How will the OCaml compiler react to the following definition: `let convert (x : int) : float = float_of_int x;;`?

- [ ] It will report a *syntax error*.
- [ ] It will report a *type error*.
- [x] It will *accept* the definition. ✔

This is valid code, although since the function does nothing more than apply another function with the same argument, we could have instead used the fact that functions are normal values in OCaml, and simply assigned the existing function to a new identifier. An experienced programmer would thus have written `let convert = float_of_int;;`.

Vous avez utilisé 1 essais sur 1