SYNTAX OF BOOLEAN OPERATIONS (2/2 points)

Select the valid expressions that evaluate to `false`.

- [ ] `! true`
- [ ] `! ! false`
- [x] `not true` ✔
- [ ] `not not false`
- [x] `not (not false)` ✔

✔

- The boolean negation is the function `not`. The operator `!` exists but has a different meaning.
- The expression `not not false`, whithout parenthesis, is read as the application of two arguments to the function `not`. If it were accepted by the typechecker, this would be compiled to a call to the function `not` passing it two arguments, namely `not` and `false`.

Select the syntactically valid expressions that do not use deprecated operators.

- [ ] `true and false`
- [x] `true && false` ✔
- [ ] `true & false`
- [ ] `false or false`

- Exceptions, input/output and imperative constructs
- Modules and data abstraction

- [x] `false || false` ✔
- [ ] `false | false`

✔

- The boolean conjunction and disjunction are written respectively `&&` and `||`.
- The single character operator `&` is deprecated; and the character `|` has a completely different meaning.
- The textual operator `or` is indeed an infix boolean operator but it has been deprecated; the keyword `and` has a different meaning.

Vous avez utilisé 2 essais sur 3

SIMPLE BOOLEAN EXPRESSIONS (2/2 points)

What is the result of compiling and evaluating `false and true`?

- (•) Syntax error. ✔
- ( ) Type error.
- ( ) `false`
- ( ) `true`

The keyword `and` is not an infix operator, and it has a completely different meaning. For the boolean conjunction you shoud use `&&`.

What is the result of compiling and evaluating `not false || true`?

- ( ) Syntax error.

- Type error.
- `false`
- `true` ✔

Function applications have a greater priority than arithmetic operators; this is read as `(not false) || true`.

Vous avez utilisé 1 essais sur 3

SIMPLE COMPARISON EXPRESSIONS (3/3 points)

What is the result of compiling and evaluating `1 < 2 && 2 <> 3`?

- Syntax error.
- Type error.
- `false`
- `true` ✔

The inequality operator is written `<>`. The operator `!=` exists but it has a different meaning.

What is the result of compiling and evaluating `1 = true`?

- Syntax error.
- Type error. ✔
- `false`

- [ ] `true`

While the comparison operators are polymorphic, you cannot compare two values of different types. Hence, the expressions `true = 1` is rejected by the typechecker.

What is the result of compiling and evaluating `1 < 2 < 3`?

- [ ] Syntax error.
- [x] Type error. ✔
- [ ] `false`
- [ ] `true`

- The comparison operators are left associative. It means that this expression is syntactically valid, and is read as: `(1 < 2) < 3`.This is not a ternary comparison, which OCaml does not have. Although this pattern can make sense in some cases, it is here the result of a (common) beginner error.
- Fortunately, the typechecker rejects this expression, because the polymorphic comparaison cannot compare two values of different types, namely `1 < 2` fo type `bool` and `3` of type `int`.

Vous avez utilisé 1 essais sur 3

SIMPLE COMPARISON EXPRESSIONS (BIS) (6/6 points)

Warning: you only have 1 attempt (but anyway the result will not count in the final grading).

What is the result of compiling and evaluating `true = 1 && not (3 = 4)`?

- Syntax error.
- Type error. ✔
- `false`
- `true`

You can't compare booleans and integers.

What is the result of compiling and evaluating `false <> 0 && 3 <= 4`?

- Syntax error.
- Type error. ✔
- `false`
- `true`

You can't compare booleans and integers.

What is the result of compiling and evaluating `true <> false && 3 <> 4`?

- Syntax error.
- Type error.
- `false`
- `true` ✔

What is the result of compiling and evaluating `not (2 >= 1) || 3 = 4`?

- Syntax error.
- Type error.
- `false` ✔
- `true`

What is the result of compiling and evaluating `(not true) = false || 3 = 4`?

- Syntax error.
- Type error.
- `false`
- `true` ✔

This is read as `((not true) >= false) || (3 = 4)`.

What is the result of compiling and evaluating `not true = false || 3 = 4`?

- Syntax error.
- Type error.
- `false`
- `true` ✔

This is read as `((not true) >= false) || (3 = 4)`.

Vous avez utilisé 1 essais sur 1