What's the main reason to add type hints to your Python program?

Answer: Type hints help you as a developer to understand and edit your own code better.

Type hints don't help increase the performance of your code, because they are actually completely ignored by the interpreter. As a result, they don't help the interpreter either to find mistakes in your code. Type hints are tools for the programmer to understand the code better, and to help avoid mistakes when for example passing arguments to functions and methods, or when working with a return value from a function or method.

In a GUI application, we keep track of the actions that a user has taken, in order to support undo and redo behavior. Each action is represented by a class which is a subclass of Action, for example: EditAction, DeleteAction, SelectAction, ReplaceAction. There's also a function get_past_actions, that return the actions the user did in the past. What's the return type of this function?

Answer: `list[Action]`

The `get_past_actions` function returns a list of actions because the order is important when undoing and redoing actions. The `Action` class provides an abstraction for all the possible actions that can be taken. This way, it's easy to add other action types (like `ClickAction` or `DragAction`) in the future without having to change the code of the action undo and redo manager.

Consider the following function:

```
def compute_average(numbers: list[float]) -> float
```

which computes the average of a list of floating point numbers. Now consider this assignment:

```
a = compute_average
```

What's the type of a?

Answer: `Callable[[list[float]], float]`

We're storing the reference to the function `compute_average` in the variable `a`. Note that we don't call the function `compute_average` here. We're just storing the reference. so, `a` is a function reference which has the type `Callable`. When you define a callable type in Python, you provide a list containing the types of the arguments (which is in another list) and the return type, like so:

```
Callable[[argument_type_1, argument_type_2, ...], return_type]
```

There's one argument of type `list[float]` so the first position in the list contains a list with one value in it: `list[float]` . The return type is `float` so that's in the second position of the list.

Which of the following statements is true (select one or more answers)?

The `int` type in Python can represent an arbitrarily large number (limited only by the memory of the machine the code is running on). *True: integer numbers are indeed unlimited in size. Python dynamically changes the memory needed to store the integer number as needed (which actually makes using integers slow for larger numbers).*

If you don't add type hints to your code in Python, it will not run. *False: Type hints are not required for Python to run a program. They are just a way to make the code more readable.*

Regular strings and formatted strings (f-strings) have the same type: `str` . *True: formatted strings are simply a tool to build strings with variables and expressions more easily. In the end, a formatted string is also a string.*

Since a type hint in Python is only a hint, it's not bad practice to pass a value of a different type instead. *False: if you use types, be strict with them. Types are meant to be a helpful tool to you, but that only works if you adhere to them.*

If you pass a value of a type that's different from the expected type (defined using a type hint), the code will not run. *False: type hints are ignored by the Python interpreter. So even if your linter flags a type hint error, the code will still run. The interpreter will just ignore the type hint and use the type of the value passed to the function or method.*

A function in Python always has the `Callable` type. *True: any function is indeed always a Callable!*

When is typing information gathered in a dynamically typed language?

Answer: at runtime.

A language is dynamically typed if the type is associated with run-time values. The types are inferred automatically, saving you time as a developer because you won't need to write all the type annotations.

How can we classify the type system that Python uses?

Answer: dynamically and strongly typed.

Python is dynamically typed: types are inferred automatically at run-time. Python is also strongly typed: Python enforces restrictions on mixing values with different types (i.e. the expression `5 + '3'` is not allowed in Python).

Why does the expression `"Hello" + 20` result in an error in Python?

Answer: Python is strongly typed.

The type of the left hand side of the expression is `str`, and the type of the right hand side is `int`. Mixing values of different types is not allowed in Python, meaning it is strongly typed.

In Python, you're allowed to assign values of different types to the same variable:

```
a = 4
a = True
```

What does this say about Python's type system?

Answer: Python is dynamically typed.

Since the type is inferred at run-time, you can assign values of different types to the same variable.