```
f = (s => s.toLowerCase());
```

causes the function expression to be processed without a contextual type, now inferring ‘s’ and the result of the function to be of type Any as no type annotations are present.

In the following example

```
interface EventObject {
    x: number;
    y: number;
}

interface EventHandlers {
    mousedown?: (event: EventObject) => void;
    mouseup?: (event: EventObject) => void;
    mousemove?: (event: EventObject) => void;
}

function setEventHandlers(handlers: EventHandlers) { ... }

setEventHandlers({
    mousedown: e => { startTracking(e.x, e.y); },
    mouseup: e => { endTracking(); }
});
```

the object literal passed to ‘setEventHandlers’ is contextually typed to the ‘EventHandlers’ type. This causes the two property assignments to be contextually typed to the unnamed function type ‘(event: EventObject) => void’, which in turn causes the ‘e’ parameters in the arrow function expressions to automatically be typed as ‘EventObject’.