```rust
macro_rules! impl_to_perf_string_on_to_string {
    ($($t:ty), *) => {
        $(
            impl ToPerfString for $t {
                fn to_perf_string(&self) -> String {
                    self.to_string()
                }
            }
        )*
    };
}

/// Let’s you simply create a resource from multiple metrics. It’s a bit like the vec! macro.
/// ```rust
/// # #[macro_use]
/// # extern crate nagiosplugin;
/// #
/// # use nagiosplugin::{SimpleMetric, State};
/// #
/// # fn main() {
/// let m1 = SimpleMetric::new("test", Some(State::Ok), 12, None, None, None, None);
/// let m2 = SimpleMetric::new("other", None, true, None, None, None, None);
/// let resource = resource![m1, m2];
/// # }
/// ```
#[macro_export]
macro_rules! resource {
    ($( $m:expr ), *) => {
        {
            use $crate::Resource;
            let mut r = Resource::new(None, None);
            $(
                r.push($m);
            )*
            r
        }
    };
}

macro_rules! metric_string {
    ($name:expr, $( $tps:expr), *) => {
        {
            let mut s = String::new();
            s.push_str(&format!("{}=", $name));
            $(
                s.push_str(&$tps.to_perf_string());
                s.push(’;’);
            )*
            s.trim_end_matches(’;’).to_string()
        }
    };
}

#[cfg(test)]
mod tests {
    use crate::SimpleMetric;

    #[test]
    fn test_resource_macro() {
        let m1 = SimpleMetric::new("test", None, 12, None, None, None, None);
        let m2 = m1.clone();

        let _resource = resource![m1.clone()];
        let _resource = resource![m1.clone(), m2.clone()];
    }
}
```