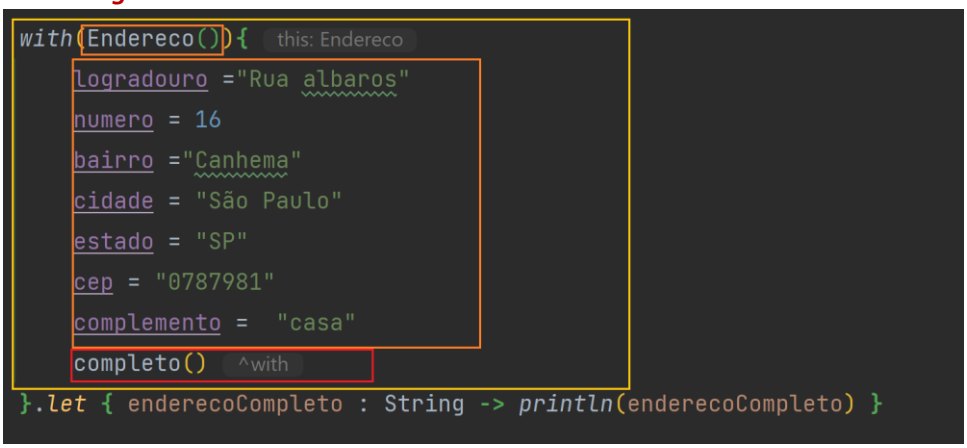
Scope functions são basicamente funções que tem um objeto no escopo delas. Por exemplo, quando chamamos o ***"let"*** a partir de um **objeto**, esse **objeto é inserido no escopo do let:**

A ideia principal das **funções de escopo** é **reduzir verbosidade ao se trabalhar com um objeto**. Em **Java a gente teria que instanciar o objeto por uma referência e ir chamando comportamentos através da referência repetindo-a sempre**, no let do Kotlin é bem menos verboso.

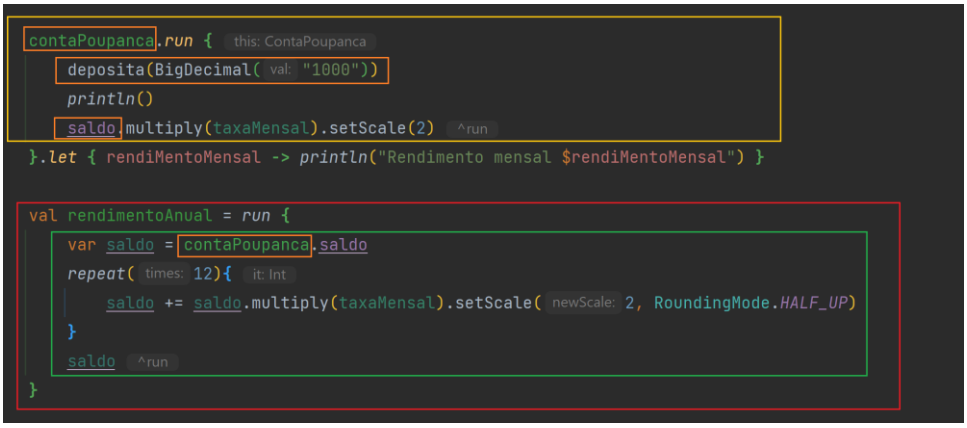
Existem **5 tipos de funções de escopo** *("let", "run", "apply", "also" e "with")* no **Kotlin**, basicamente elas fazem a mesma coisa com **algumas diferenças no retorno delas**. **Apply** e **also retornam** o próprio **Objeto** que está no contexto, as **demais retornam** o **resultado da lambda.**

Para diferenciar quando usar cada uma delas uma boa dica é se atentar na ***semântica da função***, elas são bem descritivas.

- ***Let : ele*** se torna uma **função de extensão do objeto**, bom para ***trabalhar (se encadear)*** com **resultado de uma ou mais funções,** bom para tratar **valores nulos** e criar **variáveis com escopo limitado.**
- ***With:*** não é uma função de extensão, a ideia dele é **receber o objeto como argumento** e **trabalhar com ele diretamente**. Semanticamente dá pra ler como ***"com esse objeto, faça esse código":***



- ***Run:*** tem a proposta do “let” mas é similar ao “with”. Ele é útil quando você precisa além de inicializar um objeto também retorna o valor da operação com ele. **(também dá pra usar sem um objeto/extensão):**



- **Apply:** a ideia do apply é usar o objeto de contexto para modifica-lo e devolver ele mesmo. É algo parecido com ***“aplique essas mudanças nesse objeto e devolva ele”:***
- ***Also:*** ele não interfere com o estado do objeto, é como se fosse um ***“Aliás, faz tal coisa”*** depois ou antes de alguma operação.