R-documentatie

1. Introductie

In deze R-documentatie staat hoe de regels voor de documentatie toegepast moeten worden. Per onderwerp wordt omschreven aan welke voorwaarden de R-code dient te voldoen. Over elk onderwerp zal er een korte uitleg volgen welke regels hier zullen worden gehanteerd.

Deze R-documentatie is samengesteld uit 3 bronnen. De opbouw en de basis van deze R-documentatie is bijna gelijk aan de Java-documentatie, maar daarbij zijn de `SkeletonScriptR.R` en de Google's R Style Guide erin verwerkt. Wanneer er regels in Google's R Style Guide haaks staan op regels in de Java-documentatie, krijgen de regels van de Java-documentatie de prioriteit.

2. Documentatie van de scripts in R

Opbouw van de code:

De documentatie van de code van een script moet als volgt zijn opgebouwd.

1. De naam van het R-script en de namen van de variabelen en functies in een script moeten normale Nederlandse of Engelse namen zijn.
   Het is dus niet toegestaan rare afkortingen of dergelijke te gebruiken voor variabelen en functies.
2. De scriptnaam moet het doel van de functie van het script zelf representeren.
3. Het R-script moet eindigen met de extensie `.R`.
4. De variabelen en functies in dat script moeten iets te maken hebben met het doel van de functie van het script zelf.

Opbouw van het commentaar:

De documentatie van het commentaar van een script moet als volgt zijn opgebouwd.

1. Onderaan het script moet een korte omschrijving staan van het doel van het script zelf en wat zijn functie is.
   Zie hiervoor het sketetonscript `SkeletonScriptR.R`.
2. Bij elke functie staat een korte omschrijving van het doel van die functie zelf en wat zijn taak is.

3. Regelgeving voor programmeren en coderen

3.1. Opbouw van een R-script:

De opbouw van een R-script is het skeletscript van R: `SkeletonScriptR.R`

Deze opbouw ziet er globaal als volgt uit. In cursieve tekst kan er wat extra opmerkingen worden gegeven.

1. Import-statements door middel van: `library([packagenaam])`
   Het aanroepen van andere R-scripts door middel van: `source([R-script])`
2. Globale variabelen met hun globale data
   Let op: Deze variabelen moeten elk een '`g_`' aan het begin van hun variabelenaam hebben staan.
   Elke nieuwe variabele moet op een nieuwe regel komen te staan.
3. Functie `showUsageInformation()`
   Dit is 1 functie en kan worden aangeroepen door `-h`, `--h`, `-help` of `--help` als 1e command line argument mee te geven, als je het R-script opstart.
4. Alle functies van het werkzame deel van het R-script
   Tussen elke functie moet een witte regel komen. Overloaded functies dienen bij elkaar gegroepeerd te worden.
5. Functie `main`

In de `main`-functie is het de bedoeling dat alleen wordt bepaald wat de command line arguments zijn en de functies van het werkzame deel van het R-script kunnen aangeroepen worden.
Indien er een bepaalde bewerking of berekening gebeurt in de `main`-functie, dient er daarvoor een aparte nieuwe werkzame functie gemaakt te worden.

6. Korte omschrijving van het doel en de functie van het script

3.2. Lettertype:

Het lettertype dat voor de code gehanteerd zal moeten worden, is `Courier New`. `Courier New` heeft hierbij als voordeel dat elke karakter even hoog en even breed is in pixels. Hierdoor is code makkelijker leesbaar.

3.3. Coderen:

Een regel code mag bestaan uit ongeveer maximaal 79 karakters. Er zijn uitzonderingen:

- Links van sites
- Package en import statements
- Command line arguments die gecopy-pastet zijn.

Als een regel code toch langer wordt dan 79 karakters, moeten na het 79e karakter de volgende regel(s) met een indent van 4 spaties beginnen.

Functies mogen uit hooguit ongeveer 15 regels code bestaan, exclusief de commentaarregels en de declaratie van de globale variabelen.

3.4. Assignment:

Wanneer een primitief datatype of object wordt gekoppeld aan een variabelenaam, of wanneer een functie wordt aangemaakt, dient de assignment ervan altijd te gebeuren met de zogenaamde assignmentpijl (`<-`) en niet met het =-teken (=).

Let op dat je bij een assignment aan globale variabelenamen altijd de volgende assignmentpijl dient te gebruiken: `<<-`

4. Formatting

4.1. Block-like constructs:

Block-like constructs worden altijd tussen accolades geschreven. Accolades worden gebruikt bij de commando's `if`, `else`, `for` en functies.

Als voorbeeld wordt hieronder een functie gebruikt.

```
doeIets <- function(input) {
    print(input)
}
```

De openingsaccolade (`{`) moet achter de titel van de methode staan met 1 spatie ertussen. De afsluitingsaccolade (`}`) moet helemaal op het laatst gezet worden en dient direct onder dezelfde positie te staan, als waar de block-like construct begint. De nieuwe regels code, die tussen de accolades komen, moeten een indent hebben van 4 spaties. Dit geldt voor iedere keer, als weer een nieuwe block-like construct gemaakt wordt.

Meteen op de nieuwe regel moet de code daadwerkelijk beginnen. Er mag geen regel worden overgeslagen. Elk nieuwe statement code is een nieuwe regel code. Dus 2 regels code achter elkaar op 1 regel met een '`;`'

ertussen is niet toegestaan. De code die komt na de afsluitingsaccolade, moet direct op een nieuwe regel beginnen met een blanco regel ertussen. Uitzonderingen zijn `if-else`-structuren.

De `if-else`-structuren moeten worden gecodeerd op een andere manier. Aan de rode regels zie je achter de afsluitingsaccolade direct de `else`. Voorbeeld is hieronder weergegeven.

```
if (conditie) {
    Statement1
} else {
    Statement2
}
```

4.2. Regels overslaan:

Bij de volgende situaties wordt er een regel overgeslagen tussen regels code:
1. Tussen de ene en de daarop volgende functie
2. Om regels code te groeperen als elke groep een andere taak doet.

4.3. Spaties:

In de volgende situaties moet er een spatie geplaatst worden:

1. Om een gereserveerd woord te verdelen, zoals `if, for` na een open paranthesis.
2. Om een gereserveerd woord te verdelen, zoals `else` na een afsluitingsaccolade.
3. Voor elke openingsaccolade.

In alle andere gevallen om zinnen en commando's van elkaar te verdelen.

4.5. Becommentariëren van de code:

Code hoort altijd becommentarieerd te worden. Zo moet sowieso elke functie in het script de code van commentaar zijn voorzien.

4.5.1. Becommentariëren van een functie in het script:

Bij elke functie moet een omschrijving gegeven worden van wat die functie precies doet en waarom hij gemaakt is. Dit dient geschreven te worden onder de functietitel en de openingsaccolade, maar voor de 1e regel code in de block-like construct.

Het is belangrijk dat je in dat commentaar beschrijft wat elke variabele is en waar het voor dient en dat je beschrijft wat de returndata zijn van je functie.

4.5.2. Wijze van becommentariëren:

Bij het becommentariëren van de code moet er gebruik gemaakt worden van de volgende schrijfwijze:

```
# Commentaar 1.
# Commentaar 2.
```

Dit is de manier hoe het script becommentarieerd moet worden.

4.5.3. Horizontal alignment bij becommentariëren:

Dit wordt gedaan bij het commentaar naast `#.` Hierbij wordt het commentaar direct naast een regel code 1 spatie verder geschreven.

Voorbeelden:

```
a <- 5 # a is gelijk gesteld aan 5.
docent <- "Martijn" # Bapgc-docent is Martijn.
```

4.6. Naamgeving:

4.6.1. Naamgeving van referentievariabelen:

Naamgevingen van variabelen gaan volgens het LowerCamelCase-principe met de volgende opbouw:
`[datatype][mooie naam voor variabele]`

4.6.2. Afkortingen referentievariabelen:

Voor de afkortingen van soorten datatypen is er een overzicht en uitleg ervan gegeven in deze sectie.
Let op: Hier staat omschreven welke afkortingen gebruikt moeten worden voor de datatypen.

Wanneer er geprogrammeerd wordt, is het heel belangrijk de referentievariabelen naar objecten een naam te geven met een systematische opbouw.

Let op: Hoe referentievariabelen hieronder worden benoemd, is een manier en absoluut geen verplichting!

Hieronder volgt een systematische opbouw van een referentievariabele.

`[afkorting voor het type object][omschrijving van het object]`

Hierbij moet je bij de referentievariabele, waarbij meerdere afzonderlijke woorden zijn gebruikt elk woord erin beginnen met een hoofdletter. Deze opbouw geldt ook voor functies.

De afkortingen voor de soorten datatypes zijn in onderstaande tabellen weergegeven.

Tabel 1: Afkortingen voor datatypes van opslagelementen

Lijsten van datatype / klassetype:	**Afkorting(en):**
Array	`arr`
Dataframe	`df`
Factor	`fct`
List	`lst`
Matrix	`mt`
Tabel	`tbl`
Vector	`vct`

Hieronder volgt nog een aantal voorbeelden van referentievariabelen, waarbij de naamgeving aan de hand van bovenstaande tabellen wordt gehanteerd.

Voorbeelden van referentievariabelen:

Voorbeeld 1: Je maakt de lege lijst Landen aan.

```
lstLanden <- list()
```

Voorbeeld 2: Je maakt de matrix Inhoud van 5 bij 3 aan met daarin de getallen van 1 t/m 15.

```
mtInhoud <- matrix(c(1:15), nrow=5, ncol=3)
```

Voorbeeld 3: Je maakt de vector Getallen aan met de volgende getallen erin: 4, 3, 64, 3, 98

```
vctGetallen <- c( 4, 3, 64, 3, 98)
```

4.6.3. Naamgevingen van functienamen:

Naamgevingen van methodenamen gaan volgens het LowerCamelCase-principe:

Voorbeelden:

```
berekenTotaal <- function(punten1, punten2) {
bepaalCyclus <- function(data) {
```

Wanneer functies worden aangeroepen, moet in hun parameterlijst eerst de argumenten worden meegegeven zonder standaardwaarden en daarna pas argumenten met standaardwaarden.
Meerdere regels aan argumenten voor de parameterlijsten van de functies zijn toegestaan, maar een argument mag alleen gebroken worden op de komma's. Bij een nieuwe regel moet er worden ingesprongen met een indent van 4 spaties.

Voorbeelden:

```
bepaalScore <- function(naam, afstand, query, eigenschap, aantalDagen,
    aantalUren, aantalSeconden, show.plot = TRUE)
bepaalScore <- function(naam, afstand, query, eigenschap, aantalDagen,
    aantalUren, aantalSeconden, show.plot = TRUE, show.graphicsOn = FALSE,
    legendOn = TRUE)
```

4.6.4. Naamgevingen van scripts:

Naamgevingen van scripts (`.R`) zijn geschreven volgens het UpperCamelCase-principe.

Voorbeelden:

```
BerekenenVerrijking.R
BepalenKwaliteitenReads.R
```

4.6.5. Naamgevingen van globale variabelen:

Wanneer een variabelenaam een globale variabele betreft, moet er voor zijn naamgeving een '`g_`' worden geplaatst.

Voorbeelden:

```
g_vctGetallen;
g_lstWaarden;
```

4.7. Initialisaties:

4.7.1. Initialisatie van een array:

Wanneer een array moet worden geïnitialiseerd, moet de volgende opbouw worden aangehouden:

```
arrGetallen <- array([range getallen])
```

Hieronder volgt een aantal voorbeelden.

Voorbeeld 1: Een 1-dimensionale array maken met de getallen 1 t/m 50 erin:

```
arrGetallen <- array(1:50)
```

Voorbeeld 2: Een 3-dimensionale array maken met getallen 1 t/m 24 met de dimensies 3, 4 en 2:

```
arrGetallen <- array(1:24, dim=c(3,4,2))
```

Wanneer de bovenstaande array met inhoud langer wordt dan 79 karakters, moet er naar de volgende regel gegaan worden met een indent van 4 spaties. Wanneer er meerdere regels nodig zijn, is het niet nodig bij die extra regels extra indents toe te passen.

Bronnen

1. Java-documentatie
2. Google's R Style Guide
   https://google.github.io/styleguide/Rguide.xml
3. Skeleton Script R