**SCOPE**

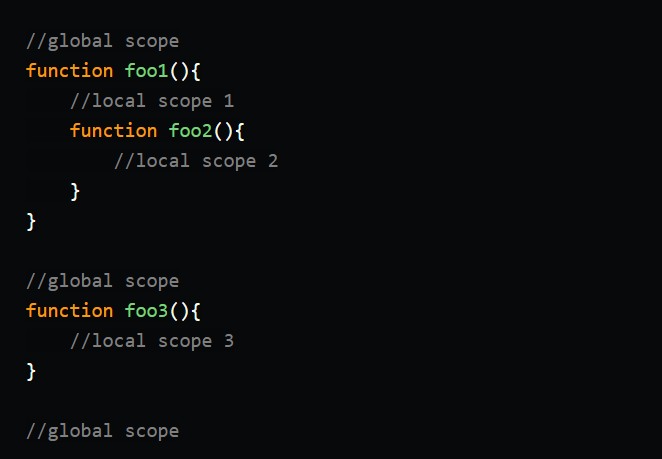
**(Global Scope)**

JavaScript փաստաթղթում կա միայն մեկ global scope : Բոլոր գործառույթներից դուրս գտնվող տարածքը համարվում է global scope, և global scope ներսում սահմանված փոփոխականները կարող են մուտք գործել և փոփոխվել ցանկացած այլ scope-ում:

**(Local Scope)**

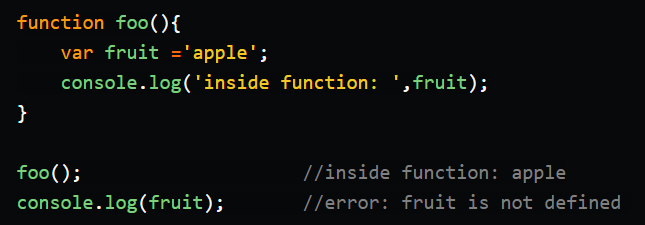
Ֆունկցիաների ներսում հայտարարված փոփոխականները դառնում են Local ֆունկցիայի համար և դիտարկվում են համապատասխան լոկալ տիրույթում: Յուրաքանչյուր ֆունկցիա ունի իր scope: Նույն փոփոխականը կարող է օգտագործվել տարբեր ֆունկցիաներում, քանի որ դրանք կապված են համապատասխան ֆունկցիաների հետ և փոխադարձ տեսանելի չեն:

Var փոփոխականը տարբեր կերպ է վարվում ֆունկցիաների scope-ում և բլոկների scope-ում: function scope-ում var-ով հայտարարված փոփոխականը չի կարող մուտք գործել այդ function scope-ից դուրս:

****

**(Function Scope)**

* Ֆունկցիայի ներսում հայտարարված փոփոխականը հասանելի չէ ֆունկցիայի սկոպից դուրս։
* JavaScript-ում յուրաքանչյուր ֆունկցիա ստեղծում է նոր scope:
* Var, let և const-ով հայտարարված փոփոխականները միանգամայն նման են ֆունկցիայի ներսում հայտարարվելիս:
* JavaScript ֆունկցիայի մեջ հայտարարված փոփոխականները դառնում են LOCAL այդ ֆունկցիայի համար:
* Այս փոփոխականները հեռացվում են հիշողությունից, երբ ֆունկցիան ավարտվում է, ուստի փոփոխականների անունները կարող են կրկին օգտագործվել որոշ այլ ֆունկցիաներում:
* Function Scope-ում հայտարարված փոփոխականները դառնում են անհասանելի և հեռացվում են, երբ ավարտվում է ֆունկցիայի աշխատանքը, և պահպանվում են հիշուղությունում։ Այդ պատճառով էլ այդ փոփոխականները դառնում են հասանելի, երբ ֆունկցիան կանչվում է և դառնում են անհասանելի երբ ավարտվում է ֆունկցիայի աշխատանքը։



**(Block Scope)**

* Մինչև ES6-ը (2015), JavaScript-ն ուներ միայն Global Scope և Function Scope: ES6-ը ներկայացրեց երկու կարևոր JavaScript հիմնաբառեր՝ let և const: Այս երկու հիմնաբառերը ապահովում են Block Scope JavaScript-ում:
* { } բլոկի ներսում հայտարարված փոփոխականները չեն կարող մուտք գործել բլոկ դրսից.
* Var-ով հայտարարված փոփոխականները չեն կարող ունենալ Block Scope:

