

INSTITUTO FEDERAL
Rio Grande do Sul

Campus
Farroupilha

Paradigmas de Linguagens de Programação

Tarefa 6

Docente: Professor Cléber Macieski – Tecnólogo em Análise e Desenvolvimento de Sistemas
cleber.macieski@farroupilha.ifrs.edu.br

Discente(s): Luciano Magri
lucianomagri.std@gmail.com

Prazo: 16/11/2022		Peso: 20 % N2
-------------------	--	---------------

Análise de linguagens quanto a escopo

Analise 2 linguagens de programação a sua escolha e crie um breve relatório sobre os pontos vistos da aula 13. O relatório deve conter imagens de exemplos de código simples para ilustrar as características.

		1/12
--	--	------

Escopo de variáveis nas linguagens de programação Python e Bash

		2/12
--	--	------

Introdução

As primeiras linguagens de programação não tratavam as variáveis e funções através de escopos, o que significa que todas as variáveis eram globais e estavam acessíveis de qualquer parte do código. Porém essa abordagem torna difícil manter o controle do valor de cada variável, pois se espera que uma variável tenha determinado valor em certo momento da execução, mas ela pode ter sido alterada por outra parte do software antes.

Escopo de uma variável se refere ao link entre essa variável com a região do código onde ela é válida e pode ser utilizada ou referenciada. Em outra parte do código, um entidade ou variável com o mesmo nome irá se referir para outro endereço de memória, representado uma entidade diferente. Ou seja, o escopo é a região do programa onde um item como: uma variável, uma contante, uma função, etc. tem um nome identificador que é reconhecido.

Diferentes linguagens de programação tratam o escopo de forma diferentes, algumas contem palavras reservadas que modificam o escopo das variáveis e funções. Porém de forma forma geral, as linguagens pode ser divididas em dois grandes grupos, linguagens de escopo léxico ou estático e linguagens de escopo dinâmico.

Em linguagens de escopo estático, o escopo da variável é definido pela estrutura do código, e não será modificado posteriormente. Enquanto no escopo dinâmico, o mesmo é modificado de acordo com as chamadas de função em tempo de execução.

Desenvolvimento

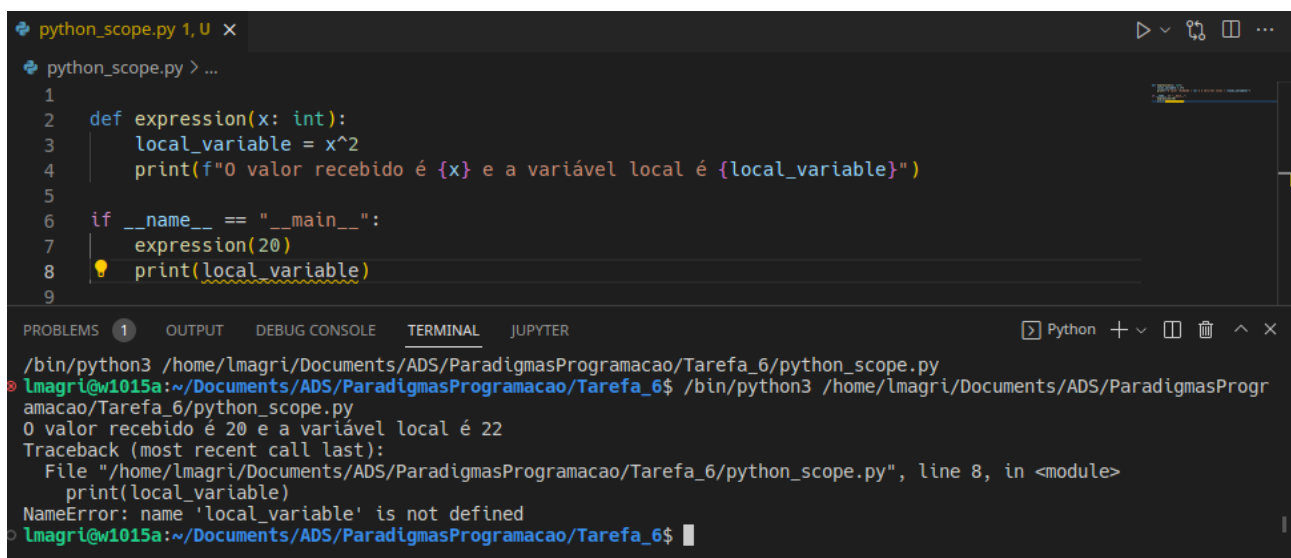
Tanto as linguagens de escopo estático, quanto as linguagens de escopo dinâmico dispõem de formas para indicar se uma variável será acessível de qualquer parte do código (escopo global) ou apenas em um espaço de código (escopo local). E cada linguagem implementa isso de forma diferente.

1. Escopo de variáveis em Python

Python utiliza escopo estático com o conceito de regra LEGB, que representam *Local*, *Enclosing*, *Global* e *Built-in*:

- *Local*: é o bloco ou corpo de qualquer função ou expressão lambda e contém os nomes das variáveis que serão visíveis apenas dentro da função.
- *Enclosing (non local)*: é um escopo especial que apenas existe em funções aninhadas, pois as variáveis de uma função interna tem acesso às variáveis da função externa.
- *Global (módulo)*: é o escopo de mais alto nível do programa ou *script*, e contém todos os nomes definidos neste nível que são acessíveis para todos os níveis inferiores.
- *Built-in*: é um escopo especial que é criado sempre que se roda um programa ou script. Esse escopo contém os nomes, funções e atributos que são padrão da linguagem e também estão disponíveis para qualquer parte do código.

O programa ou *script* irá procurar pelos nomes das variáveis em uma ordem determinada: *local*, *enclosing*, *global*, *built-in*, respectivamente. Caso o nome exista em um escopo, a busca é interrompida e o valor é utilizado, caso o nome não exista em nenhum dos escopos e apresentado um erro.



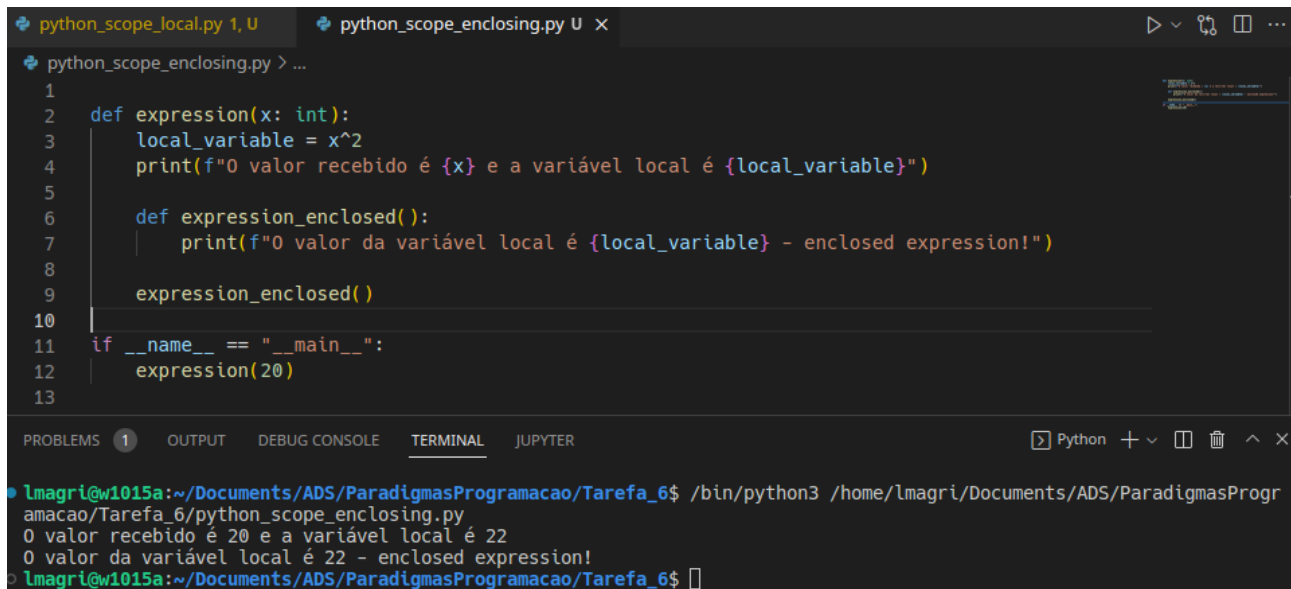
```
python_scope.py 1, U x
python_scope.py > ...
1
2 def expression(x: int):
3     local_variable = x^2
4     print(f"0 valor recebido é {x} e a variável local é {local_variable}")
5
6 if __name__ == "__main__":
7     expression(20)
8     print(local_variable)
9

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
/bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgramacao/Tarefa_6/python_scope.py
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ /bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgramacao/Tarefa_6/python_scope.py
0 valor recebido é 20 e a variável local é 22
Traceback (most recent call last):
  File "/home/lmagri/Documents/ADS/ParadigmasProgramacao/Tarefa_6/python_scope.py", line 8, in <module>
    print(local_variable)
NameError: name 'local_variable' is not defined
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 1 - Exemplo variável local não acessível no escopo global [Fonte: O autor].

Como pode ser observado na Imagem 1 acima, a variável com nome “variavel_local” retorna um erro ao tentar ser acessada do escopo global (a razão do *if statement* representar o escopo global será elaborado mais a frente).

Já na Imagem 2, abaixo, podemos verificar que a variável “variavel_local” pode ser acessada pela função interna “expression_enclosed”, isso ocorre porque a referida função está no *enclosing scope* da função “expression”, não gerando nenhum erro na execução.



```
python_scope_local.py 1, U python_scope_enclosing.py U X
python_scope_enclosing.py > ...
1
2 def expression(x: int):
3     local_variable = x^2
4     print(f"0 valor recebido é {x} e a variável local é {local_variable}")
5
6     def expression_enclosed():
7         print(f"0 valor da variável local é {local_variable} - enclosed expression!")
8
9     expression_enclosed()
10
11 if __name__ == "__main__":
12     expression(20)
13

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Python + - [] X
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ /bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgr
amacao/Tarefa_6/python_scope_enclosing.py
0 valor recebido é 20 e a variável local é 22
0 valor da variável local é 22 - enclosed expression!
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 2 - Exemplo variável no escopo especial *enclosing* [Fonte: O autor].

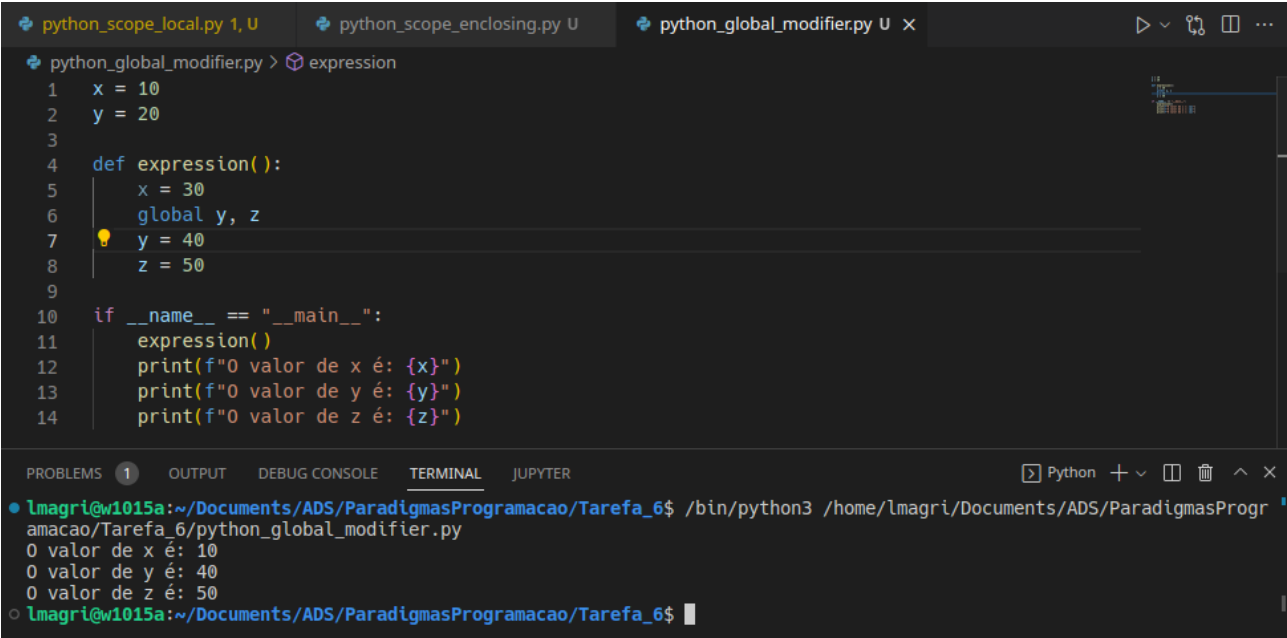
Ao executar um programa em Python, o interpretador cria um módulo para rodar o programa, a entrada do programa será no escopo do módulo que será o escopo global do programa. O nome dado ao escopo global do módulo em Python é “__main__”. Por esta razão, é considerado uma boa prática de programação em Python iniciar a função principal do programa verificando se o escopo é o “__main__”, desta forma, se o seu módulo estiver sendo executado por outro programa, a função principal será executada somente quando chamada explicitamente, evitando sua execução no momento em que o módulo é importado. Conforme Imagem 2, acima, a função “expression” só será executada se “python_scope_enclosing” for o escopo global em execução.

Isso traz a pergunta, e a variável `__name__`? A mesma não está declarada em nenhum lugar do código, como o interpretador sabe o que é essa variável se a mesma não foi declarada? A resposta para essas perguntas está no escopo *built-in*. “__name__” é uma variável declarada nesse escopo e será criada sempre que um script vai ser executado pelo interpretador Python, e é reservada para o nome do módulo (nome do arquivo .py). Porém, caso o arquivo seja o arquivo principal, ou *main*, o interpretador irá atribuir a string “__main__” para essa variável.

1.1 Modificadores de escopo

Em Python existem duas palavras reservadas para modificar o comportamento padrão de atribuição de escopo para variáveis, “*global*” e “*nonlocal*”. A palavra reservada *global* serve para indicar que uma variável faz parte do escopo global, mesmo que esteja sendo declarada em um escopo local, ou para atribuir um novo valor à uma variável global ao invés de criar uma nova variável local.

A Figura 3, abaixo, mostra a utilização da palavra reservada *global* em Python. Observa-se que as variáveis “x” e “y” foram declaradas no escopo global com os valores de 10 e 20 respectivamente, enquanto a variável “z” foi declarada apenas no escopo da função “expression”. Porém, após a execução da função o valor de “x” no escopo global continua sendo 10, enquanto o valor de “y” foi modificado pela função no escopo global para 40, enquanto o valor de 50 atribuído à “z” pode ser acessado no escopo global.

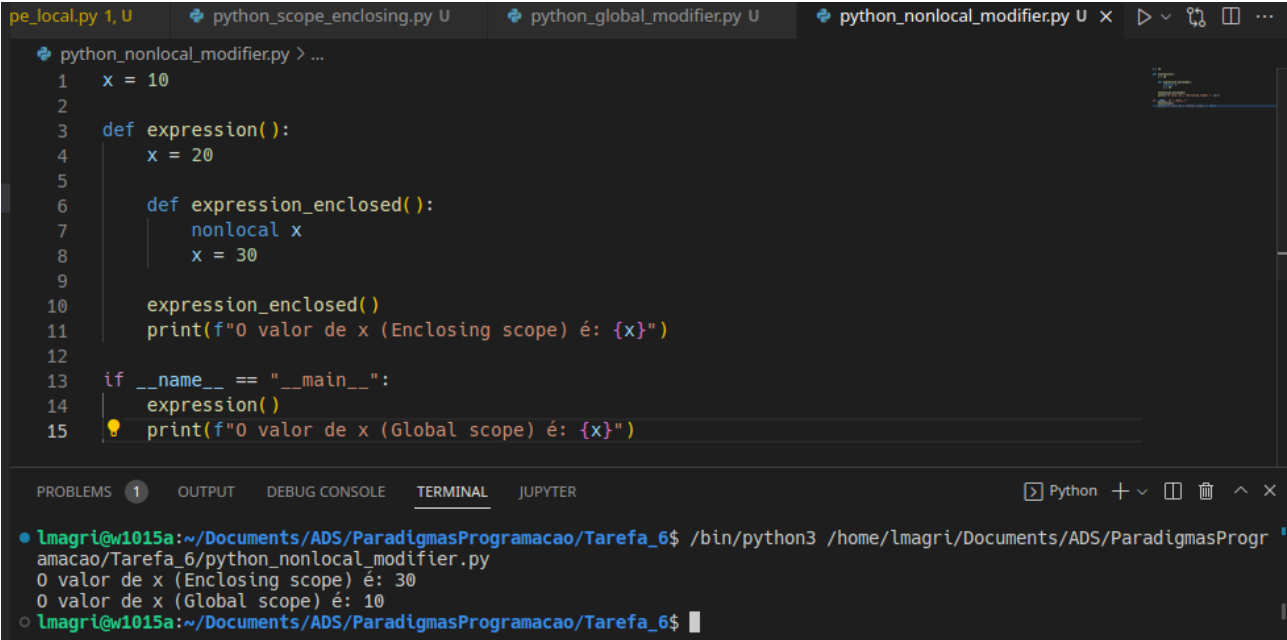


```
python_global_modifier.py > expression
1 x = 10
2 y = 20
3
4 def expression():
5     x = 30
6     global y, z
7     y = 40
8     z = 50
9
10 if __name__ == "__main__":
11     expression()
12     print(f"O valor de x é: {x}")
13     print(f"O valor de y é: {y}")
14     print(f"O valor de z é: {z}")
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ /bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgr
amacao/Tarefa_6/python_global_modifier.py
O valor de x é: 10
O valor de y é: 40
O valor de z é: 50
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 3 - Exemplo de utilização da palavra reservada *global* [Fonte: O autor].



```
python_nonlocal_modifier.py > ...
1 x = 10
2
3 def expression():
4     x = 20
5
6     def expression_enclosed():
7         nonlocal x
8         x = 30
9
10    expression_enclosed()
11    print(f"O valor de x (Enclosing scope) é: {x}")
12
13 if __name__ == "__main__":
14     expression()
15     print(f"O valor de x (Global scope) é: {x}")
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ /bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgr
amacao/Tarefa_6/python_nonlocal_modifier.py
O valor de x (Enclosing scope) é: 30
O valor de x (Global scope) é: 10
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 4 - Exemplo de utilização da palavra reservada *global* [Fonte: O autor].

A palavra reservada *nonlocal*, por sua vez, é utilizada para referenciar que a variável que se quer modificar é a variável do *enclosing scope*, como pode ser observado na saída do programa

1.2 Relação dos atributos com herança de classes

No exemplo apresentado na Imagem 5, abaixo, a classe “C” é construída utilizando herança das classes “A” e “B”, respectivamente, e ao executarmos o método “print_x” em uma instancia da classe, verificamos que o valor de “x” é 10, ou seja, o valor declarado na classe “A”. Como o interpretador escolheu o valor 10 ao invés de 20, declarado na classe “B” ?

Imagem 5 - Exemplo herança de variáveis por classes [Fonte: O autor].

Para acessarmos a variável “x” referenciada na super classe “A” através da classe “C”, conforme programa da Imagem 6, será necessário acessar explicitamente da forma “A.x”, isso pode ser realizado no método `__init__` da classe “C”.

```
python_heranca.py > ...
1 class A:
2     x = 10
3
4 class B:
5     x = 20
6
7 class C(B, A):
8     def __init__(self):
9         super().__init__()
10
11     def print_x(self):
12         print(self.x)
13
14 if __name__ == "__main__":
15     c = C()
16     c.print_x()
```

```
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ /bin/python3 /home/lmagri/Documents/ADS/ParadigmasProgramacao/Tarefa_6/python_heranca.py
20
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 6 - Exemplo herança de variáveis por classes trocando posição das super classes [Fonte: O autor].

2. Escopo de viáveis em Bash

Em Bash, o escopo das variáveis é dinâmico. Apesar de ainda existir o conceito de variáveis locais e globais, isso é definido dinamicamente durante a execução do *script*. Isso significa que uma variável local pode ser acessada por qualquer função que seu escopo chamar, se tornando uma “variável global” para a escopo de uma função que foi chamada pelo escopo inicial.

Para representar essa situação, podemos verificar na Imagem 7, abaixo, que o valor da variável “x” é alterado para “two” após a execução da função “expression_two” mesmo dentro do escopo da função “expression” e com a variável declarada como local.

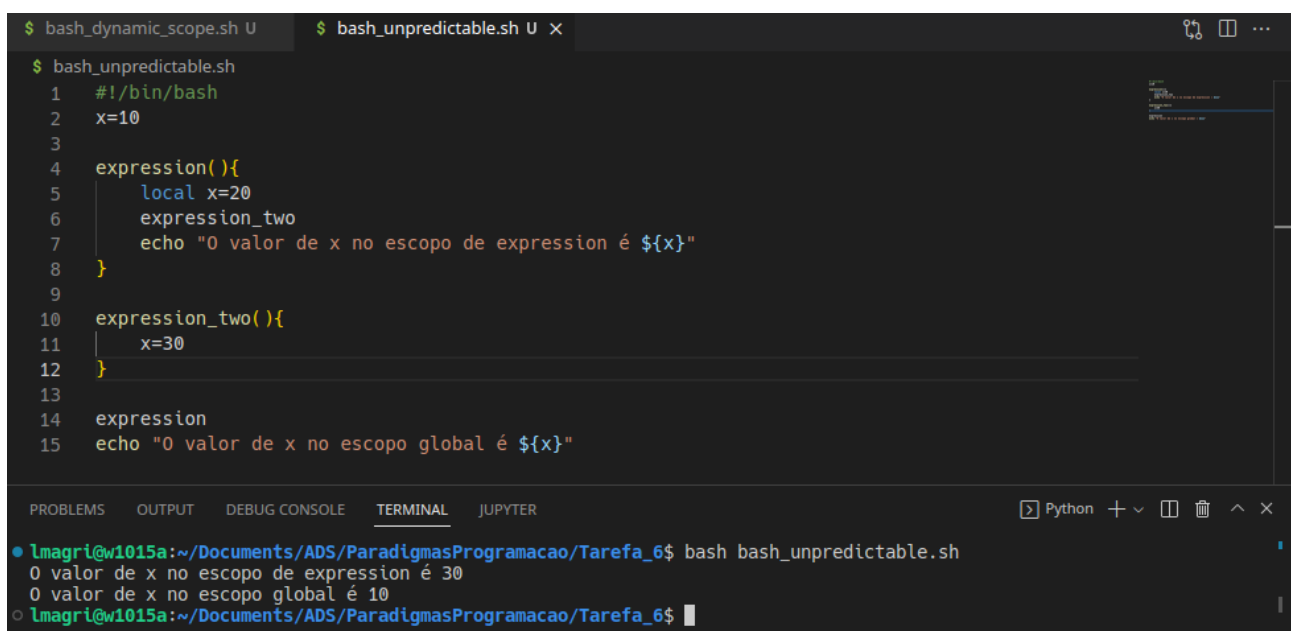
```
$ bash_dynamic_scope.sh U X
$ bash_dynamic_scope.sh
1 #!/bin/bash
2
3 expression(){
4     local x=one
5     expression_two
6     echo "0 valor de x agora é ${x}"
7 }
8
9 expression_two(){
10     x=two
11 }
12
13 expression
```

```
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ bash bash_dynamic_scope.sh
0 valor de x agora é two
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 7 - Script em Bash demonstrando escopo dinâmico da variável “x” [Fonte: O autor].

A maioria das linguagens de programação modernas utilizam escopo estático, pois com escopo dinâmico, o valor da variável vai depender da execução e não será possível saber exatamente seu valor antes disso. Para exemplificar, a Imagem 8, abaixo, apresenta uma variável “x” declarada no escopo global de um programa em Bash com valor atribuído de 10, e outra variável “x” declarada como local no escopo da função “expression” com valor atribuído de 20. E então uma terceira função “expression_two” que modifica “x” para 30. É fácil pensar que na execução do código demonstrado na Imagem 8, o valor de “x” no escopo global seria alterado para 30 na execução da função “expression_two”, porém, se observa que o “x” alterado foi o do escopo local de “expression”. Isso se deve ao fato que “expression_two” foi chamada dentro do escopo de “expression”.

O programa abaixo é extremamente simples e é fácil entender o valor de “x”, mesmo com escopo dinâmico. Porém, conforme o programa cresce e existem múltiplas chamadas de cada função que podem ser executadas pelos usuários em ordens arbitrárias, isso pode se tornar um problema para manutenção do código e correção de *bugs*. Uma saída para isso em linguagens de escopo dinâmico é declarar todas as variáveis como locais e recebê-las via argumentos nas funções.



```
$ bash_dynamic_scope.sh U $ bash_unpredictable.sh U x
$ bash_unpredictable.sh
1  #!/bin/bash
2  x=10
3
4  expression(){
5      local x=20
6      expression_two
7      echo "O valor de x no escopo de expression é ${x}"
8  }
9
10 expression_two(){
11     x=30
12 }
13
14 expression
15 echo "O valor de x no escopo global é ${x}"

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ bash bash_unpredictable.sh
O valor de x no escopo de expression é 30
O valor de x no escopo global é 10
lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 8 - Script em Bash demonstrando escopo dinâmico da variável “x” [Fonte: O autor].

2.1 Modificadores de escopo em Bash

Em Bash os modificadores de escopo são: *local*, *global*, *env*, *export*. Os últimos dois são implementados a nível do emulador *shell* do sistema GNU. Para referencia, *env* declara uma variável de ambiente do sistema GNU, enquanto *export* torna a variável disponível em outros processos do Bash que estejam rodando no sistema operacional.

Os modificadores local e global, funcionam da da mesma forma que em Python, com as respectivas ressalvas da linguagem ser de escopo dinâmico.

```
$ bash_dynamic_scope.sh U    $ bash_unpredictable.sh U    $ bash_local_global.sh U X
$ bash_local_global.sh
1  #!/bin/bash
2
3  expression(){
4      local y=50
5      declare -g z=40
6  }
7
8  expression
9  echo "0 valor de x no escopo global é: ${y}"
10 echo "0 Valor de z pode ser acessado do escopo global: ${z}"

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
Python + - □ □ ^ X

● lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$ bash bash_local_global.sh
0 valor de x no escopo global é:
0 Valor de z pode ser acessado do escopo global: 40
○ lmagri@w1015a:~/Documents/ADS/ParadigmasProgramacao/Tarefa_6$
```

Imagem 9 - Script em Bash demonstrando variável *local* e *global* [Fonte: O autor].

Em Bash, não será exibido erro de execução caso uma variável não tenha sido declarada ou não tenha sido atribuído um valor para ela. O que acontecerá é que o interpretador irá atribuir uma *string* em branco para valor da variável. Isso responde porque não existe valor para a primeira linha do programa mostrado na Imagem 9, acima. Como “y” é uma variável local da função “expression” o valor impresso é “” (*empty string*). A variável “z”, por sua vez, foi declarada como global, e pode ser acessada fora do escopo da função onde foi declarada.

Conclusão

Conforme apresentado, as linguagens de programação modernas tentem a utilizar escopos estáticos ou léxicos, pois, apesar de serem mais complexos de implementar, promovem um ambiente mais amigável e mais previsível para o programador.

A possibilidade de definir o escopo de uma variável através da leitura do código, permite que as IDEs (*Integrated development Environment*) possam definir com precisão se uma variável foi declarada para aquele escopo e indicar problemas no código, antes mesmo de executá-lo. Além disso, programas escritos em linguagens de escopo estático são mais fáceis de debugar.

Por sua vez, o escopo dinâmico pode ser útil em linguagens de *scripting* como Bash, por sua facilidade de implementação e menor complexidade dos programas que serão executados através do interpretador.

		11/12
--	--	-------

Referências:

Programming Fundamentals. Disponível em:< <https://press.rebus.community/programmingfundamentals/chapter/scope/> >. Acesso em: 15 nov. 2022.

Programming Language Concepts/Binding and Scope. Bilgisayar Mühendisliği, 2008. Disponível em:< https://ocw.metu.edu.tr/pluginfile.php/2982/mod_resource/content/0/lectures/06-binding.pdf >. Acesso em: 15 nov. 2022.

Python Variable Scope – Local, Global, Built-in, Enclosed. DataFlair Team, 2018. Disponível em:< <https://data-flair.training/blogs/python-variable-scope/> >. Acesso em: 15 nov. 2022.

Python Scope & the LEGB Rule: Resolving Names in Your Code. Leodanis Pozo Ramos (Realpython). Disponível em:< <https://realpython.com/python-scope-legb-rule/> >. Acesso em: 15 nov. 2022.

Approach Bash Like a Developer - Part 20 - Scoping. Binaryphile, 2018. Disponível em:< <https://www.binaryphile.com/bash/2018/09/01/approach-bash-like-a-developer-part-20-scoping.html> >. Acesso em: 15 nov. 2022.

Bash Functions – Declaration, Scope, Arguments, etc. Hitesh J, 2019. Disponível em:< <https://www.websertvertalk.com/bash-functions> >. Acesso em: 15 nov. 2022.

Bash variables — Things that you probably don’t know about it. Matheus Lozano, 2021. Disponível em:< <https://medium.com/unboxing-the-cloud/bash-variables-things-that-you-probably-dont-know-about-it-8a5470887331> >. Acesso em: 15 nov. 2022.

3.5.3 Shell Parameter Expansion. GNU.org. Disponível em:< https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html >. Acesso em: 15 nov. 2022.

		12/12
--	--	-------