Atividade

Cada grupo ficará responsável pela implementação de três programas em três diferentes paradigmas. O **grupo 1** ficará com as atividades **marcadas com 1 em cada paradigma** e assim por diante.

Grupo 5

Nomes:

Laura Silveira Pinzon
Matheus dos Santos Viegas
Paulo Ricardo Dalsoto Junior
Pedro Felipe Paulino Rosinha
Tomás Dalpiaz Strieder

Interpretador utilizados:
Scheme: https://ideone.com https://ideone.com/v7pZ09
Prolog: https://swish.swi-prolog.org/
Python:
https://onecompiler.com/python?gad_source=1&gad_campaignid=7520626229&gbraid=0AAAAACuMHwOx-0FTQh3Uwcu1dQpE-HeOg&gclid=Cj0KCQjw953DBhCyARIsANhIZoYiDYWl-DTFkQzjJhjdyH2Ix4bh3RMsr7pN1GTq_VIi7HE6kt6W1BUaAicBEALw_wcB
+
JetBrains

Paradigma Funcional (Scheme, recursão, funções puras, imutabilidade)

5. Classificador de números: Categorizar números em pares, ímpares e primos usando filtros e funções matemáticas.

```
(define (checkDivisor n i)
  (cond ((> i (sqrt n)) #t)                    ; i > Vn -> é primo
        ((= (remainder n i) 0) #f)             ; se n % i == 0 (tal que o número só pode ser divisível por 1
                                                ; e ele mesmo e a recursão começa com 2)
                                                ;-> não é primo
        (else (checkDivisor n (+ i 1)))))      ; recursão com i++ (próx. divisor)

(define (ehPrimo? n)
  (checkDivisor n 2)) ;Chama a função auxiliar de testar divisor

(define (ehPar? x)
  (= (remainder x 2) 0)) ; Retorna x % 2 == 0

(define (ehImpar? x)
  (not (= (remainder x 2) 0))) ; Retorna x % 2 != 0

(define (classificarNumeros numeros)
  (define pares (filter ehPar? numeros))
  (define impares (filter ehImpar? numeros))
  (define primos (filter ehPrimo? numeros))
  (list pares impares primos)) ;Retorna uma list com 3 listas "filtradas"

(classificarNumeros '(1 2 3 4 5 6 7 8 9 14120 1121 13 15)) ;' indica que é uma lista de argumentos
```


```
(define (checkDivisor n i)
  (cond ((> i (sqrt n)) #t)                    ; i > Vn -> é primo
        ((= (remainder n i) 0) #f)             ; se n % i == 0 (tal que o número só pode ser divisível por 1
                                                ; e ele mesmo e a recursão começa com 2)
                                                ;-> não é primo
        (else (checkDivisor n (+ i 1)))))      ; recursão com i++ (próx. divisor)

(define (ehPrimo? n)
  (checkDivisor n 2)) ;Chama a função auxiliar de testar divisor

(define (ehPar? x)
  (= (remainder x 2) 0)) ; Retorna x % 2 == 0

(define (ehImpar? x)
  (not (= (remainder x 2) 0))) ; Retorna x % 2 != 0

(define (classificarNumeros numeros)
  (define pares (filter ehPar? numeros))
  (define impares (filter ehImpar? numeros))
  (define primos (filter ehPrimo? numeros))
  (list pares impares primos)) ;Retorna uma list com 3 listas "filtradas"
```

(classificarNumeros '(1 2 3 4 5 6 7 8 9 14120 1121 13 15)) ;' indica que é uma lista de ;argumentos

;para output visual usar ‘write ((...))’

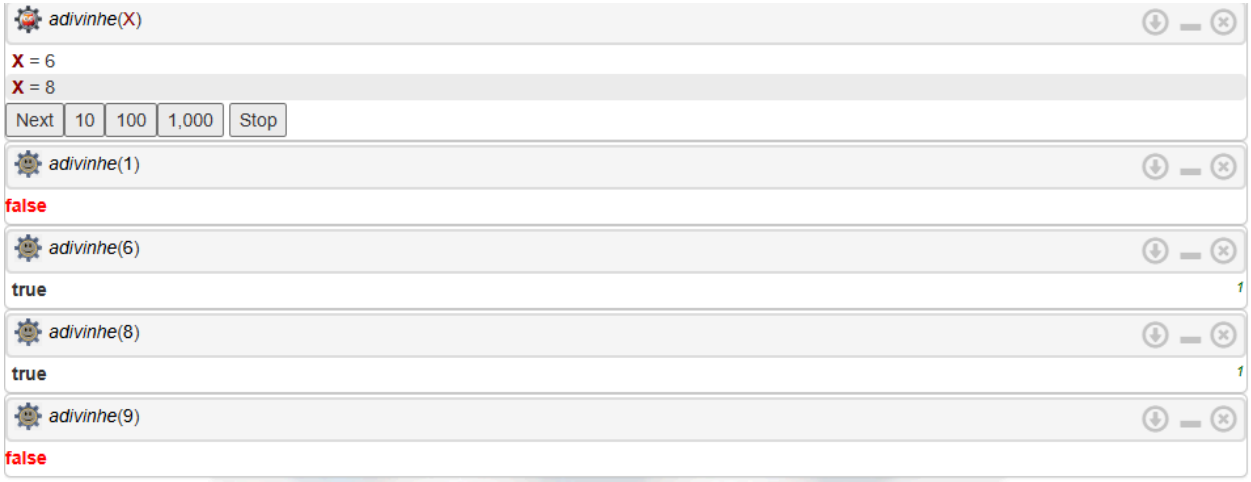
Paradigma Lógico (Prolog, fatos, regras, consultas)
5. Jogo "Adivinhe o Número" com restrições lógicas: Modelar dicas e possibilidades como restrições para encontrar o número correto.

```
% Base de fatos: possíveis números de 1 a 10
numero(1).
numero(2).
numero(3).
numero(4).
numero(5).
numero(6).
numero(7).
numero(8).
numero(9).
numero(10).

% Regras para dicas/restrições
maior_que(N, X) :- numero(X), X > N.
menor_que(N, X) :- numero(X), X < N.
par(X) :- numero(X), 0 is X mod 2.
impar(X) :- numero(X), 1 is X mod 2.
multiplo_de(M, X) :- numero(X), 0 is X mod M.
entre(A, B, X) :- numero(X), X >= A, X =< B.

% Regra principal para deduzir o número com base nas restrições
adivinhe(X) :-
    numero(X),
    maior_que(5, X),
    par(X),
    menor_que(9, X).
```

QUERY PARA O SWISH:



Paradigma Imperativo / Orientado a Objetos (Python, C#, controle explícito)
5. Conversor de bases numéricas iterativo: Realizar conversão entre bases usando laços e manipulação direta dos dígitos.

```
class BaseConverter:
    def __init__(self):
        self.digits = "0123456789ABCDEF"

    def convert_to_decimal(self, number: str, base: int) -> int:
        number = number.upper()
        decimal = 0
        power = 0

        for digit in reversed(number):
            value = self.digits.index(digit)
            if value >= base:
                raise ValueError(f"Digit '{digit}' is invalid for base {base}")
            decimal += value * (base ** power)
            power += 1
        return decimal

    def convert_from_decimal(self, decimal: int, base: int) -> str:
```

```
        if decimal == 0:
            return "0"

        result = ""
        while decimal > 0:
            remainder = decimal % base
            result = self.digits[remainder] + result
            decimal //= base
        return result

    def convert(self, number: str, from_base: int, to_base: int) -> str:
        decimal = self.convert_to_decimal(number, from_base)
        return self.convert_from_decimal(decimal, to_base)


if __name__ == "__main__":
    converter = BaseConverter()

    while True:

print("\n╔═══════════════════════════════════════════════════════════════════════════════╗\n║
Number Base Converter")
        num = input("║ Enter the number (or 'QUIT' to quit): ")
        if num.lower() == "quit":
            break

        try:
            from_base = int(input("║ From base (2-16): "))
            to_base = int(input("║ To base (2-16): "))
            if not (2 <= from_base <= 16) or not (2 <= to_base <= 16):
                print("║ Bases must be between 2 and 16.")
                continue

            result = converter.convert(num, from_base, to_base)
            print(f"║ {num} (base {from_base}) → {result} (base {to_base})")

        except ValueError as e:
            print("Error:", e)


print("╚═══════════════════════════════════════════════════════════════════════════════╝")
```

```
class BaseConverter:
    def __init__(self):
        self.digits = "0123456789ABCDEF"

    def convert_to_decimal(self, number: str,
base: int) -> int:
        number = number.upper()
        decimal = 0
        power = 0

        for digit in reversed(number):
            value = self.digits.index(digit)
            if value >= base:
                raise ValueError(f"Digit
'{digit}' is invalid for base {base}")
            decimal += value * (base ** power)
            power += 1
        return decimal

    def convert_from_decimal(self, decimal: int,
base: int) -> str:
        if decimal == 0:
            return "0"

        result = ""
        while decimal > 0:
```

```
            remainder = decimal % base
            result = self.digits[remainder] +
result
            decimal //= base
        return result

    def convert(self, number: str, from_base:
int, to_base: int) -> str:
        decimal = self.convert_to_decimal(number,
from_base)
        return self.convert_from_decimal(decimal,
to_base)


if __name__ == "__main__":
    converter = BaseConverter()

    while True:

print("\n╔═══════════════════════════════════════
═══════════════╗\n║ Number Base Converter")
        num = input("║ Enter the number (or
'QUIT' to quit): ")
        if num.lower() == "quit":
            break

        try:
```

```
            from_base = int(input("╠ From base
(2-16): "))
            to_base = int(input("╠ To base
(2-16): "))
            if not (2 <= from_base <= 16) or not
(2 <= to_base <= 16):
                print("╠ Bases must be between 2
and 16.")
                continue

            result = converter.convert(num,
from_base, to_base)
            print(f"╠ {num} (base {from_base}) →
{result} (base {to_base})")

        except ValueError as e:
            print("Error:", e)


print("╚═══════════════════════════════════════════
═══════════╝")
```