SEVEN
publicações acadêmicas

Linguagem natural, pensamento computacional e desenvolvimento cognitivo

https://doi.org/10.56238/sevened2023.006-061

Marcelo Magalhães Foohs
Doutor em Informática na Educação. Professor Associado do Departamento de Estudos Especializados da Universidade Federal do Rio Grande do Sul.
ORCID: https://orcid.org/0000-0002-4735-0732
E-mail: 00145282@ufrgs.br

RESUMO
Neste artigo, explora-se a linguagem de programação como manifestação do pensamento computacional, a partir da tradução de abstrações em códigos, para que os computadores as compreendam. São abordados alguns conceitos-chave, como sintaxe, abstração e resolução de problemas, destacando sua relação com as funções psicológicas superiores. Analisa-se a interação sinérgica entre o pensamento computacional e a linguagem natural, examinando como essa simbiose pode influenciar a coesão e a coerência textual. Apresenta-se uma sequência didática que faz uso dessa sinergia, com critérios de avaliação somativa, em que a remidiação é a peça-chave para a transposição de narrativas linguísticas para o mundo digital, empregando a plataforma Scratch como uma ferramenta catalisadora. Isso visa não apenas enriquecer a expressão criativa dos estudantes, mas também integrar, de maneira eficaz, a linguagem natural, o pensamento computacional e a tecnologia educacional.

Palavras-chave: Pensamento computacional, Linguagem natural, Funções psicológicas superiores, Programação, Tecnologia educacional.

1 INTRODUÇÃO

No cenário educacional brasileiro, os testes oficiais têm destacado lacunas preocupantes no desenvolvimento de habilidades cruciais para os estudantes. Não são meras estatísticas, mas revelam profundas deficiências na produção textual, na interpretação de informações e na resolução de problemas complexos. Essas lacunas refletem um desafio maior: a falta de uma abordagem integrada que una os poderes da linguagem, a lógica do pensamento computacional e as funções cognitivas superiores.

Este capítulo se propõe a explorar a intersecção multifacetada entre linguagem de programação, linguagem natural e funções cognitivas superiores, visando não somente identificar essas lacunas, mas oferecer soluções visionárias para preenchê-las. Apoiado em fundamentos teóricos de renomados estudiosos como Ingedore Koch (2020), com sua profunda análise da linguística textual, e nas ideias de Vygotsky (2001) sobre funções cognitivas superiores, este trabalho busca estabelecer uma conexão íntima entre conceitos aparentemente distintos, mas intrinsecamente ligados.

A linguagem de programação, longe de ser apenas uma ferramenta de comunicação entre humanos e computadores, é o meio pelo qual traduzimos abstrações, lógica e algoritmos para uma forma que os computadores não apenas compreendem, mas executam. É o código da mente

computacional, permitindo aos programadores expressarem sua visão lógica e conceberem soluções para problemas complexos.

Nesta jornada de compreensão, aspectos-chave serão explorados minuciosamente: sintaxe, abstração e resolução de problemas. Será destacada sua intrínseca relação com as funções psicológicas superiores, analisando-se como a interação sinérgica entre o pensamento computacional e a linguagem natural pode influenciar a coesão e a coerência textual.

Além disso, propõe-se uma abordagem educacional que integra esses elementos de maneira eficaz, utilizando uma sequência didática que se baseia nessa simbiose. A remidiação assume um papel-chave na transposição de narrativas linguísticas para o mundo digital, incluindo critérios de avaliação somativa e empregando a plataforma Scratch como uma ferramenta catalisadora. O objetivo não é apenas enriquecer a expressão criativa dos estudantes, mas integrar de maneira eficaz a linguagem natural, o pensamento computacional e a tecnologia educacional para o desenvolvimento cognitivo dos alunos.

Este capítulo não é apenas uma exposição teórica, mas um convite à reflexão sobre como a sinergia entre esses domínios pode moldar um futuro educacional promissor para o Brasil. Estamos diante de uma oportunidade única: a oportunidade de redefinir o paradigma educacional, capacitando nossos alunos a inovar, criar e solucionar problemas complexos de forma eficaz para os desafios do século XXI.

2 PROCEDIMENTOS METODOLÓGICOS

A abordagem metodológica adotada neste capítulo segue um processo estruturado, visando proporcionar uma compreensão abrangente e consistente dos temas apresentados. Dividimos a estrutura do capítulo em seções distintas, cada uma delas focando em elementos específicos e interligados do pensamento computacional e sua relação com a linguagem natural.

O capítulo segue uma organização lógica, iniciando com uma introdução abrangente sobre o pensamento computacional e sua relação com linguagem de programação e linguagem natural. A partir daí, cada subseção foi estruturada para explorar um conceito-chave do pensamento computacional, relacionando-o diretamente com exemplos práticos em linguagem Python. A metodologia adotada envolveu uma sequência de demonstrações, utilizando a linguagem de programação como veículo para ilustrar e conectar os conceitos abordados.

A escolha dos exemplos foi baseada na sua capacidade de elucidar os conceitos teóricos de forma clara e acessível. Cada exemplo foi detalhadamente explicado, enfatizando as relações entre o pensamento computacional e os elementos da linguagem natural, com o intuito de tornar o conteúdo mais tangível e aplicável.

A construção progressiva dos exemplos, de subseção para subseção, permite uma exploração gradual e aprofundada dos temas, promovendo a compreensão progressiva do conteúdo e evidenciando as interações entre os conceitos apresentados.

Ao longo do desenvolvimento, buscamos constantemente contextualizar os exemplos no cenário educacional, demonstrando como esses conceitos podem ser aplicados no processo de aprendizagem, visando fortalecer habilidades cognitivas e o desenvolvimento integral dos estudantes.

Essa metodologia foi adotada com o propósito de oferecer uma estrutura didática, facilitando a compreensão e a aplicabilidade dos conceitos apresentados, além de promover uma visão integrada entre o pensamento computacional, a linguagem natural e o desenvolvimento cognitivo.

3 LINGUAGEM DE PROGRAMAÇÃO: CÓDIGO DA MENTE COMPUTACIONAL

A linguagem de programação é o meio pelo qual os seres humanos traduzem suas abstrações, lógica e algoritmos de forma que os computadores possam compreendê-los e executá-los. Essencialmente, ela é o código da mente computacional, permitindo que, por meio das estratégias do pensamento computacional, os programadores expressem sua visão lógica e criem soluções para problemas complexos.

A definição de pensamento computacional, elaborada conjuntamente pela *International Society for Technology in Education* (ISTE)[1] e *Computer Science Teachers Association* (CSTA)[2] (OPERATIONAL..., 2011), oferece uma descrição abrangente e precisa do tema, que inclui as seguintes características: 1. Formulação de problemas; 2. Organização lógica e análise de dados; 3. Representação de dados por meio de abstrações; 4. Automação de soluções por meio de pensamento algorítmico; 5. Identificação, análise e implementação de soluções eficientes; 6. Generalização e transferência para uma variedade de problemas.

Na sequência, será explorada a forma como a linguagem de programação torna-se uma ferramenta para a manifestação do pensamento computacional.

3.1 SINTAXE E ESTRUTURAÇÃO LÓGICA

Assim como na linguagem natural, onde regras gramaticais e semânticas estruturam nossas comunicações, a linguagem de programação também opera com uma sintaxe específica. Essa sintaxe é a base estrutural que permite aos programadores expressar algoritmos, estruturas de controle e lógica de programação de forma organizada e compreensível para os computadores. Vamos explorar como isso se manifesta em Python, uma linguagem conhecida por sua clareza e expressividade.

[1] Sociedade Internacional de Tecnologia na Educação (ISTE).
[2] Associação de Professores de Ciência da Computação (CSTA).

A sintaxe em Python é notável por sua simplicidade e legibilidade. Utiliza uma abordagem que valoriza a clareza do código, permitindo expressar operações complexas de forma direta. A seguir, apresentaremos exemplos que demonstram como a estruturação lógica e a sintaxe específica em Python facilitam a expressão de algoritmos e operações matemáticas de maneira concisa.

No primeiro exemplo, a sintaxe do Python, como o uso do operador de atribuição "=", o operador de adição "+" e a função "print" para exibição, permite expressar de forma clara e direta uma simples operação matemática de soma.

```
# Soma de dois números
numero1 = 10
numero2 = 5
soma = numero1 + numero2
print("A soma é:", soma)
```

No segundo exemplo, utilizamos a sintaxe do Python para calcular a média de três números. A estruturação lógica e a sintaxe específica possibilitam a realização dessa operação aritmética de maneira concisa e compreensível.

```
# Cálculo da média de três números
numero1 = 15
numero2 = 20
numero3 = 10
media = (numero1 + numero2 + numero3) / 3
print("A média é:", media)
```

Por fim, demonstramos a utilização da estrutura condicional em Python para verificar se um número é par ou ímpar. A sintaxe específica para estruturas de decisão permite controlar o fluxo do programa de acordo com condições lógicas.

```
# Verificação de número par ou ímpar
numero = 7
if numero % 2 == 0:
print("O número é par.")
else:
print("O número é ímpar.")
```

Esses exemplos ilustram como a sintaxe em Python possibilita a organização estruturada de algoritmos e lógica de programação, permitindo expressar operações de maneira eficiente e legível.

3.2 ABSTRAÇÃO E MODULARIZAÇÃO

Uma das facetas fundamentais do pensamento computacional reside na capacidade de abstrair conceitos complexos em componentes mais simples e reutilizáveis. A linguagem de programação oferece ferramentas para essa abstração, permitindo a criação de funções e objetos que representam ideias ou operações abstratas. Vamos alguns exemplos para compreender mais profundamente esse conceito.

A abstração, nesse contexto, permite encapsular lógicas ou operações específicas em estruturas reutilizáveis, como funções ou objetos, promovendo uma compreensão mais simplificada e abstrata do problema. Em um primeiro exemplo, temos uma função simples em Python que calcula o quadrado de um número. A função "calcular_quadrado" abstrai a operação matemática necessária para encontrar o quadrado de um número, tornando-a uma unidade modular e reutilizável.

```
# Função para calcular o quadrado de um número
def calcular_quadrado(numero):
return numero ** 2
# Chamando a função e imprimindo o resultado
numero = 8
print("O quadrado é:", calcular_quadrado(numero))
```

No segundo exemplo a seguir, demonstramos a criação de uma classe em Python, representando um produto. Essa classe abstrai as propriedades e comportamentos comuns de um produto, como nome e preço, facilitando a criação de instâncias específicas e reutilização de código.

```
# Criação de uma classe em Python para representar um produto
class Produto:
def __init__(self, nome, preco):
self.nome = nome
self.preco = preco
def exibir_produto(self):
print(f"Produto: {self.nome}, Preço: R${self.preco}")
# Criando uma instância da classe Produto e exibindo seus detalhes
produto1 = Produto("Caneta", 2.50)
produto1.exibir_produto()
```

Adicionalmente, apresentamos uma função que verifica se um número é positivo, negativo ou zero. Essa função encapsula a lógica de verificação e retorna um resultado abstrato baseado no número fornecido.

```
# Função para verificar se um número é positivo, negativo ou zero
def verificar_numero(numero):
```

```
if numero > 0:
return "Positivo"
elif numero < 0:
return "Negativo"
else:
return "Zero"
# Chamando a função e imprimindo o resultado
num = -7
print("O número é:", verificar_numero(num))
```

Esses exemplos destacam como a abstração e modularização em Python permitem encapsular lógicas e operações em unidades reutilizáveis, promovendo a reutilização de código e facilitando a compreensão de conceitos complexos em termos mais simples e abstratos.

3.3 RESOLUÇÃO DE PROBLEMAS E A LÓGICA

A capacidade da linguagem de programação em expressar a resolução de problemas por meio de sequências de instruções lógicas é fundamental. Ao criar algoritmos e estruturas de controle, como loops e condicionais, os programadores traduzem sua capacidade de raciocínio lógico em ações computacionais. Vamos explorar alguns exemplos que evidenciam essa relação entre resolução de problemas e a lógica em Python.

A lógica na programação permite a criação de sequências de comandos que orientam o comportamento do programa de acordo com condições específicas, refletindo o raciocínio lógico humano em ações computacionais. Em um primeiro exemplo, demonstramos o uso de uma estrutura condicional para identificar o maior número entre dois valores. A lógica da estrutura "if-else" permite a comparação de variáveis e a tomada de decisão baseada em condições lógicas.

```
# Identificação do maior número entre dois valores
numero1 = 15
numero2 = 20
if numero1 > numero2:
print("O maior número é:", numero1)
else:
print("O maior número é:", numero2)
```

No segundo exemplo apresentado a seguir, utilizamos um loop "for" para imprimir os números pares de 1 a 10. O uso da estrutura de repetição permite executar uma ação repetitiva com base em condições lógicas.

```
# Loop for para imprimir os números pares de 1 a 10
```

```
for i in range(1, 11):
if i % 2 == 0:
print(i, end=" ")
```

Por fim, apresentamos uma estrutura condicional que verifica a idade para acesso a um conteúdo restrito. Essa verificação baseada em condições demonstra a lógica por trás de decisões condicionais em programas.

```
# Verificação de idade para acesso a um conteúdo restrito
idade = 17
if idade >= 18:
print("Acesso concedido ao conteúdo restrito.")
else:
print("Você não tem idade suficiente para acessar este conteúdo.")
```

Esses exemplos visam ilustrar como a linguagem de programação permite expressar a lógica de resolução de problemas por meio de algoritmos estruturados, utilizando estruturas condicionais e de repetição para tomar decisões e realizar ações de maneira automatizada e lógica.

4 PENSAMENTO COMPUTACIONAL, FUNÇÕES PSICOLÓGICAS SUPERIORES E EDUCAÇÃO

Além de servir como código da mente computacional, a linguagem de programação, estruturada pelas estratégias do pensamento computacional, está intrinsecamente ligada às funções psicológicas superiores, podendo promover seu desenvolvimento e aprimoramento. As funções psicológicas superiores, conforme concebidas por Vigotsky (2001) e Vigotsky, Luria e Leontiev (2010), que incluem pensamento conceitual, memória voluntária, atenção, resolução de problemas, planejamento e autorregulação, imaginação e criatividade, são importantes para o desenvolvimento cognitivo do estudante e para o enfrentamento dos desafios complexos na era digital.

Dito isso, serão apresentados, sucintamente, a seguir, alguns exemplos em linguagem Python, que exemplificam a relação entre as estratégias do pensamento computacional e as funções psicológicas superiores. Além disso, é importante salientar que Vigotsky (2001, p. 65), em seu livro *Pensamento e Linguagem*, chama a atenção para o papel fundamental da educação na relação entre a aprendizagem e o desenvolvimento cognitivo:

> A nossa segunda série de investigações centrou-se sobre as relações temporais entre os processos de ensino e o desenvolvimento das funções psicológicas que lhes correspondem. Descobrimos que o ensino geralmente precede o desenvolvimento. A criança adquire certos hábitos e qualificações num dado domínio antes de aprender a aplicá-los consciente e deliberadamente. Nunca há um paralelismo completo entre o curso do ensino e o desenvolvimento das correspondentes funções.

É interessante ressaltar como essa citação de Vigotsky (2001) reforça a importância da educação nesse processo, destacando a relação entre aprendizagem e desenvolvimento cognitivo. Os exemplos em linguagem Python a seguir, podem servir como uma ponte entre os conceitos teóricos das funções psicológicas superiores e sua aplicação prática no contexto educacional. Cada exemplo apresentado oferece uma demonstração tangível de como as estratégias do pensamento computacional se entrelaçam com habilidades cognitivas fundamentais, como resolução de problemas, planejamento e autorregulação, promovendo não apenas a compreensão teórica, mas também a aplicação concreta desses conceitos no ambiente educacional.

4.1 PENSAMENTO CONCEITUAL

O pensamento conceitual envolve a habilidade de compreender e manipular conceitos abstratos. Na programação, os estudantes aplicam estratégias do pensamento computacional ao lidar com conceitos como variáveis, funções e estruturas de controle. Vamos explorar alguns exemplos que demonstrem a aplicação prática desse pensamento na linguagem Python.

O pensamento conceitual na programação refere-se à capacidade de trabalhar com conceitos abstratos de maneira a resolver problemas e criar soluções utilizando estruturas lógicas e operações específicas. No primeiro exemplo a seguir, apresentamos uma função que calcula o volume de um cubo com base no tamanho do lado fornecido. Aqui, a manipulação do conceito geométrico de volume por meio da função abstrai a fórmula matemática, evidenciando o pensamento conceitual na aplicação direta de fórmulas matemáticas.

```
# Função para calcular o volume de um cubo
def calcular_volume_cubo(lado):
return lado ** 3
# Chamando a função e imprimindo o resultado
lado_cubo = 5
print("O volume do cubo é:", calcular_volume_cubo(lado_cubo))
```

No segundo exemplo, temos uma função que verifica se um número é positivo ou não. A manipulação do conceito de positividade por meio de uma função abstrai a lógica da verificação, mostrando o pensamento conceitual na aplicação de lógica condicional.

```
# Criação de uma função para verificar se um número é positivo
def verificar_positivo(numero):
if numero > 0:
return True
else:
return False
```

```
# Verificando se um número é positivo e imprimindo o resultado
num = -7
if verificar_positivo(num):
print("O número é positivo.")
else:
print("O número não é positivo.")
```

Por fim, apresentamos o uso de listas em Python para armazenar elementos e acessá-los por meio de índices. Essa manipulação do conceito de listas exemplifica como os estudantes podem trabalhar com estruturas de dados abstratas para manipular informações de maneira organizada.

```
# Utilização de listas para armazenar elementos e manipular dados
lista_frutas = ['Maçã', 'Banana', 'Laranja', 'Morango']
print("A terceira fruta da lista é:", lista_frutas[2])
```

Estes exemplos visam ilustrar como o pensamento conceitual se manifesta na programação, permitindo aos estudantes manipular e aplicar conceitos abstratos para resolver problemas e criar soluções em Python.

4.2 MEMÓRIA VOLUNTÁRIA

A memória voluntária refere-se à capacidade de armazenar e acessar informações de forma consciente e intencional. Na programação, a organização e estruturação de informações são fundamentais para a construção de algoritmos eficientes. Vamos explorar a seguir, exemplos que ilustrem como a programação incentiva o uso da memória voluntária, possibilitando aos aprendizes armazenar e acessar informações de maneira consciente.

A memória voluntária na programação está relacionada à habilidade de armazenar e recuperar informações de maneira intencional, utilizando estruturas de dados e conceitos específicos. Neste primeiro exemplo, demonstramos o uso de um dicionário em Python para armazenar contatos e seus emails associados. Essa estrutura permite o acesso consciente às informações por meio de chaves, exemplificando a memória voluntária na organização e acesso direto a dados.

```
# Armazenamento de contatos em um dicionário
agenda_contatos = {
'João': 'joao@email.com',
'Maria': 'maria@email.com',
'Carlos': 'carlos@email.com'
}
# Acessando o email de um contato
print("Email de João:", agenda_contatos['João'])
```

No segundo exemplo, temos variáveis que armazenam dados do estado de um jogo, como pontuação, vidas e nível atual. Essa manipulação intencional de variáveis para atualizar e exibir informações evidencia a memória voluntária na programação.

```
# Utilização de variáveis para armazenar dados de um jogo
pontuacao = 1500
vidas = 3
nivel_atual = 5
# Atualizando e exibindo a pontuação do jogador
pontuacao += 500
print("Nova pontuação:", pontuacao)
```

Por fim, apresentamos uma função que calcula o total de despesas mensais com base em uma lista de valores. O uso intencional da função para armazenar cálculos exemplifica como a memória voluntária é aplicada na reutilização de soluções para resolver problemas semelhantes.

```
# Utilização de uma função para armazenar cálculos de despesas mensais
def calcular_despesas_mensais(despesas):
total_despesas = sum(despesas)
return total_despesas
# Chamando a função e exibindo o total de despesas mensais
despesas_janeiro = [1000, 1500, 800, 2000]
total = calcular_despesas_mensais(despesas_janeiro)
print("Total de despesas em janeiro:", total)
```

Estes exemplos demonstram como a memória voluntária se manifesta na programação, permitindo aos aprendizes armazenar e acessar informações de maneira intencional para resolver problemas e organizar dados em Python. Além disso, é importante ressaltar que, assim como outras funções psicológicas superiores, a memória voluntária pode ser desenvolvida com prática e exercícios regulares.

A prática de manipulação consciente de informações na programação não apenas fortalece habilidades técnicas, mas também tem um impacto significativo no desenvolvimento cognitivo dos aprendizes. Ao praticarem a organização de dados e a solução de problemas por meio da programação, os estudantes não apenas aprimoram sua capacidade de pensamento crítico, mas também fortalecem a memória voluntária, habilidade crucial em diversos aspectos da vida.

Especificamente no contexto educacional, essa prática de desenvolvimento da memória voluntária por meio da programação pode ser altamente benéfica. Ela não se limita a aprimorar as habilidades técnicas dos estudantes, mas também contribui para a melhoria de sua capacidade de aprender e resolver problemas em disciplinas diversas.

Ao integrar o desenvolvimento da memória voluntária na programação como parte do currículo educacional, estamos não só capacitando os estudantes para lidar com desafios tecnológicos futuros, mas também fortalecendo suas habilidades cognitivas gerais. Esse tipo de abordagem educacional não apenas prepara os alunos para a era digital, mas também os equipa com ferramentas cognitivas essenciais para enfrentar desafios variados ao longo de suas vidas acadêmicas e profissionais.

4.3 ATENÇÃO E FOCO NA PROGRAMAÇÃO

A atenção é a habilidade de se concentrar em uma tarefa específica, evitando distrações. Na programação, esse aspecto é crucial, exigindo atenção sustentada para compreender e resolver problemas complexos. Vamos explorar a seguir exemplos que ilustrem como a programação fortalece a capacidade de manter a atenção e o foco.

A habilidade de manter a atenção é essencial na resolução de problemas em programação. Os aprendizes precisam focar na lógica, decompor desafios em partes gerenciáveis e desenvolver estratégias para encontrar soluções. Neste primeiro exemplo, temos um programa que verifica se um número é primo. A atenção é necessária para compreender a lógica por trás do algoritmo, seguindo cada passo da iteração para verificar se o número é divisível por algum outro número além de 1 e ele mesmo.

```
# Criando um programa que verifica se um número é primo
def verifica_primo(numero):
if numero > 1:
for i in range(2, numero):
if (numero % i) == 0:
return False
return True
return False
# Verificação se um número é primo
num = int(input("Digite um número para verificar se é primo: "))
if verifica_primo(num):
print(num, "é um número primo.")
else:
print(num, "não é um número primo.")
```

No segundo exemplo, temos um algoritmo de busca binária em uma lista ordenada. A atenção é crucial para compreender a lógica por trás da busca binária, seguir cada iteração do loop e entender como o algoritmo encontra o item desejado na lista de maneira eficiente.

```
# Elaborando um programa de busca binária em uma lista ordenada
```

```
def busca_binaria(lista, item):
baixo = 0
alto = len(lista) - 1
while baixo <= alto:
meio = (baixo + alto) // 2
chute = lista[meio]
if chute == item:
return meio
if chute > item:
alto = meio - 1
else:
baixo = meio + 1
return None
# Utilizando a busca binária
minha_lista = [1, 3, 5, 7, 9]
print("Posição do número 5:", busca_binaria(minha_lista, 5))
```

Esses exemplos mostram como a atenção é fundamental na programação. O desenvolvimento dessa habilidade não apenas ajuda os estudantes a resolverem problemas complexos na programação, mas também fortalece sua capacidade de foco e concentração, habilidades essenciais para o processo educacional. A programação não só os desafia cognitivamente, mas também os prepara para manter a atenção em tarefas desafiadoras, contribuindo positivamente para seu desenvolvimento educacional.

4.4 RESOLUÇÃO DE PROBLEMAS

A resolução de problemas é uma habilidade essencial para encontrar soluções eficazes em desafios complexos. Na programação, essa habilidade é constantemente exercitada, impulsionando o pensamento estratégico e a criatividade na busca por soluções inovadoras. Vamos explorar alguns exemplos que ilustrem essa capacidade de resolver problemas de maneira eficaz em Python.

A programação oferece um ambiente propício para o desenvolvimento da resolução de problemas. Ao enfrentarem desafios algorítmicos, os aprendizes não apenas implementam soluções, mas também aprimoram sua capacidade de pensar de forma criativa e eficiente para resolver problemas. No primeiro exemplo apresentado a seguir, temos um algoritmo de busca sequencial em uma lista. A resolução de problemas é aplicada ao criar um algoritmo eficiente para encontrar um item específico na lista, iterando por cada elemento até encontrá-lo.

```
# Exemplo 1: Implementando um algoritmo de busca sequencial em uma lista
def busca_sequencial(lista, item):
```

```
for i in range(len(lista)):
if lista[i] == item:
return i
return None
# Utilizando a busca sequencial
minha_lista = [6, 2, 8, 5, 3, 9]
print("Posição do número 5:", busca_sequencial(minha_lista, 5))
```

No segundo exemplo, temos um programa que calcula o fatorial de um número. Aqui, a resolução de problemas se reflete na criação de um algoritmo eficiente para calcular o fatorial, aplicando a lógica matemática necessária para essa tarefa.

```
# Desenvolvendo um programa que calcula o fatorial de um número
def calcular_fatorial(numero):
if numero == 0 or numero == 1:
return 1
else:
fatorial = 1
for i in range(2, numero + 1):
fatorial *= i
return fatorial
# Calculando o fatorial de um número
n = 5
print("O fatorial de", n, "é", calcular_fatorial(n))
```

Esses exemplos destacam como a resolução de problemas na programação vai além da simples implementação de código. O desenvolvimento dessa habilidade proporciona aos estudantes a capacidade de enfrentar desafios de maneira estratégica, uma habilidade valiosa para o processo educacional. Ao integrar a resolução de problemas através da programação no currículo educacional, estamos não apenas ensinando a linguagem de programação, mas também fortalecendo a capacidade dos alunos de abordar problemas complexos com criatividade e eficácia. Essa habilidade vai além do domínio técnico, sendo uma ferramenta valiosa para o desenvolvimento cognitivo e a capacidade de encontrar soluções inovadoras em diversas áreas.

4.5 PLANEJAMENTO E AUTORREGULAÇÃO

O planejamento e a autorregulação são habilidades fundamentais para programadores, pois possibilitam a elaboração de planos eficientes e a capacidade de ajustar o comportamento do código

de acordo com esses planos. A seguir vamos explorar exemplos que ilustrem como esses conceitos se aplicam na programação, usando a linguagem Python.

Na programação, planejamento significa não apenas criar um código que execute uma tarefa específica, mas também antecipar possíveis cenários e erros que podem surgir durante a execução. Autorregulação refere-se à capacidade de fazer ajustes, correções e melhorias no código, garantindo um comportamento previsível e eficiente. No primeiro exemplo apresentado, criamos um sistema de login simulado em que é possível verificar se o usuário e a senha correspondem ao registro. A aplicação do planejamento ocorre ao considerar a possibilidade de erro ao inserir um usuário inexistente, sendo essencial prever e tratar essa situação.

```
# Exemplo 1: Implementando um sistema de login com tratamento de erros
usuarios_registrados = {'usuario1': 'senha1', 'usuario2': 'senha2'}
def fazer_login(usuario, senha):
try:
if usuarios_registrados[usuario] == senha:
return True
else:
return False
except KeyError:
print("Usuário não encontrado!")
return False
# Tentando fazer login
login = fazer_login('usuario1', 'senha1')
print("Login:", login)
```

No segundo exemplo, ilustramos um programa que valida a entrada do usuário para garantir que seja um número entre 1 e 100. Aqui, a autorregulação ocorre ao incorporar mecanismos para lidar com entradas inválidas, garantindo que o programa funcione corretamente mesmo diante de erros potenciais.

```
# Desenvolvendo um programa para validar entrada de dados
def validar_entrada(numero):
try:
if int(numero) > 0 and int(numero) <= 100:
return True
else:
return False
except ValueError:
```

```
print("Por favor, insira um número válido.")
return False
# Validando a entrada do usuário
entrada_valida = validar_entrada('50')
print("Entrada válida:", entrada_valida)
```

Esses exemplos evidenciam como o planejamento e a autorregulação na programação vão além da simples codificação. Ao integrar essas habilidades no contexto educacional, além de aprender a escrever código, os estudantes também desenvolvem a capacidade de antecipar cenários, planejar soluções e corrigir erros, habilidades fundamentais para a solução de problemas complexos em diversas áreas da vida e do aprendizado.

4.6 IMAGINAÇÃO E CRIATIVIDADE

A imaginação e a criatividade são fundamentais no universo da programação, proporcionando a capacidade de conceber ideias inovadoras e explorar diversas possibilidades. Na prática da programação, os estudantes têm a oportunidade de exercitar sua imaginação, criando programas e projetos que vão além do simples funcionamento lógico. Vamos explorar a seguir alguns exemplos para ilustrar como a programação pode incentivar a imaginação e a criatividade usando a biblioteca Turtle em Python.

Na programação, a imaginação se traduz na capacidade de visualizar soluções ou conceitos, enquanto a criatividade se manifesta na maneira como essas ideias são aplicadas de forma única e inovadora. Neste primeiro exemplo apresentado a seguir, utilizamos a biblioteca Turtle para desenhar um padrão caleidoscópico colorido. O objetivo é destacar como os estudantes podem expressar sua imaginação visualmente, criando algo esteticamente atraente com base na lógica de programação.

```
# Criando um desenho com a biblioteca Turtle
from turtle import *
speed(10)
bgcolor("black")
colors = ["red", "orange", "yellow", "green", "blue", "purple"]
for x in range(360):
pencolor(colors[x % 6])
width(x / 100 + 1)
forward(x)
left(59)
done()
```

No segundo exemplo, criamos um jogo simples de clicar na tela. Aqui, a criatividade dos estudantes entra em cena ao desenvolverem a lógica por trás de um jogo interativo, explorando não apenas a capacidade de desenho visual, mas também a interação do usuário com o programa.

```
# Criando um jogo simples com a biblioteca Turtle
from turtle import *
speed(0)
bgcolor("black")
color("white")
hideturtle()
def desenhar_borda():
penup()
goto(-140, 140)
pendown()
for side in range(4):
forward(280)
right(90)
left(90)
penup()
goto(0, 0)
pendown()
setheading(0)
desenhar_borda()
write("Clique dentro da área para iniciar o jogo.", align="center", font=("Arial", 16, "normal"))
def clicar(x, y):
goto(x, y)
dot(20)
onclick(clicar)
done()
```

Esses exemplos ilustram como a programação pode ser um campo fértil para a imaginação e a criatividade dos estudantes. Ao introduzir essa abordagem na educação, não apenas estamos desenvolvendo habilidades técnicas, mas também promovendo a capacidade de resolver problemas, inovar e expressar ideias. Essas habilidades são essenciais para formar cidadãos responsáveis e engajados na sociedade atual, capacitando-os a encontrar soluções criativas para desafios complexos e a contribuir de maneira significativa para o mundo em que vivemos.

5 PENSAMENTO COMPUTACIONAL E DESENVOLVIMENTO DA LINGUAGEM NATURAL

A interação entre o pensamento computacional e o aprimoramento da linguagem natural representa uma simbiose profunda e enriquecedora. Não apenas fortalece a compreensão da linguagem, mas também influencia diretamente a maneira como estruturamos, interpretamos e utilizamos a linguagem em nossa comunicação diária. Ao aplicarmos os princípios do pensamento computacional, imergimos em um mundo de lógica, abstração e solução de problemas que tem um impacto significativo no aprimoramento da linguagem humana.

A contribuição do pensamento computacional para a linguagem natural não se limita apenas à expressão textual. Ele se estende à forma como organizamos nossos pensamentos, estruturamos nossas ideias e comunicamos informações de maneira coesa e coerente. A coesão e a coerência textual, identificadas por Beaugrande e Dressler (1981) como elementos centrais na linguagem natural, é um ponto-chave a ser explorado. Ao desvendar como o pensamento computacional se entrelaça com esses elementos, adentramos não apenas na análise de estruturas linguísticas, mas também na compreensão de como a lógica computacional pode aprimorar a clareza, a eficácia e a eficiência da comunicação escrita e falada.

No decorrer desta seção, examinaremos mais de perto como os princípios do pensamento computacional se entrelaçam com a linguagem natural, analisando como suas aplicações podem enriquecer a compreensão e a comunicação no contexto linguístico.

5.1 COESÃO: A INTERAÇÃO LÓGICA DE ELEMENTOS TEXTUAIS

A coesão textual refere-se à maneira como os elementos de um texto se conectam logicamente. No mundo da programação, o pensamento computacional é um catalisador poderoso para a coesão, exigindo que os programadores organizem seus códigos de forma lógica e estruturada. Essa prática não só é aplicável à programação, mas também é valiosa na linguagem natural, pois a habilidade de estruturar ideias de maneira lógica e sequencial é fortalecida através do pensamento computacional, resultando em uma maior coerência e clareza textual.

Explorando mais essa relação, vamos adentrar alguns exemplos que ilustram a interconexão entre o pensamento computacional e a linguagem natural, focalizando na coesão dos elementos textuais. A coesão desempenha um papel crucial tanto na programação quanto na comunicação escrita, garantindo a clareza e a compreensão do que é expresso. No primeiro exemplo apresentado a seguir, a função "calcular_media" é criada para calcular a média de uma lista de números. A lógica é essencial: somam-se os elementos da lista, calcula-se a quantidade de elementos e, finalmente, calcula-se a média.

```
# Exemplo de função para calcular a média de uma lista de números
```

```
def calcular_media(lista):
soma = sum(lista)
quantidade = len(lista)
media = soma / quantidade
return media
# Lista de números
numeros = [10, 15, 12, 18, 20]
# Chamando a função para calcular a média
media_calculada = calcular_media(numeros)
# Exibindo a média
print("A média dos números é:", media_calculada)
```

No segundo exemplo apresentado a seguir, uma frase é manipulada usando Python. A frase é dividida em palavras e, em seguida, a ordem dessas palavras é revertida, demonstrando uma manipulação textual lógica.

```
# Exemplo de manipulação de texto em Python
frase = "O pensamento computacional fortalece a coesão textual."
palavras = frase.split()  # Dividindo a frase em palavras
reverso = ' '.join(palavras[::-1])  # Revertendo a ordem das palavras
# Exibindo a frase invertida
print("Frase invertida:", reverso)
```

No terceiro exemplo apresentado abaixo, um dicionário é usado para organizar informações sobre um aluno, mostrando como a estrutura lógica é aplicada na organização de dados.

```
# Exemplo de organização de informações em um dicionário Python
aluno = {
"nome": "Maria",
"idade": 22,
"curso": "Ciência da Computação",
"notas": [8.5, 9.0, 7.8, 9.5]
}
# Exibindo informações do aluno de maneira organizada
print(f"{aluno['nome']}, {aluno['idade']} anos, do curso de {aluno['curso']}, notas: {aluno['notas']}")
```

Esses exemplos sugerem que pensamento computacional, ao exigir uma estrutura lógica e organizada, fortalece a compreensão do uso da coesão tanto na programação quanto na linguagem natural. A coesão é uma habilidade vital não apenas para o código, mas também para a clareza e

compreensão textual em nossa comunicação cotidiana, desempenhando um papel crucial no desenvolvimento de indivíduos aptos a contribuir efetivamente para a sociedade.

5.2 COERÊNCIA: A HARMONIA SEMÂNTICA NO TEXTO

A coerência textual envolve a construção de um significado global e lógico em um texto. O pensamento computacional, ao estimular a organização e a conexão lógica de conceitos, pode auxiliar na criação de textos coerentes na linguagem natural. Os programadores, ao desenvolverem algoritmos e estruturas lógicas, desenvolvem a habilidade de manter uma coerência semântica em seus códigos. Essa habilidade é transferível para a linguagem natural, resultando em textos nos quais as ideias conectam-se de maneira lógica e significativa.

Vejamos alguns exemplos dessa interconexão entre o pensamento computacional e a linguagem natural, com foco na coerência dos elementos textuais. No primeiro exemplo, a função `e_palindromo` verifica se uma palavra é um palíndromo, ou seja, se ela é lida da mesma forma de trás para frente. A coerência lógica está na comparação entre a palavra original e sua versão invertida, refletindo a necessidade de uma estrutura coerente para estabelecer o significado na linguagem natural.

```
# Exemplo de função palindromo
def e_palindromo(palavra):
return palavra == palavra[::-1]
# Verificando se uma palavra é um palíndromo
resultado = e_palindromo("reconhecer")
if resultado:
print("É um palíndromo.")
else:
print("Não é um palíndromo.")
```

No segundo exemplo, apresentado na sequência, a função `ordenar_numeros` organiza uma lista de números em ordem crescente. A coerência lógica reside na utilização do método de ordenação, garantindo que os números estejam em uma sequência lógica. Isso reflete a necessidade de organização coesa e coerente para transmitir ideias de forma lógica na linguagem natural.

```
# Exemplo função ordenar números
def ordenar_numeros(lista):
return sorted(lista)
# Ordenando uma lista de números
numeros = [8, 3, 5, 1, 9, 4]
lista_ordenada = ordenar_numeros(numeros)
print("Lista ordenada:", lista_ordenada)
```

No terceiro exemplo, a função `fibonacci` gera a sequência de Fibonacci com base no número de termos especificados. A coerência lógica está na recursividade matemática utilizada para gerar a sequência, garantindo que cada número seja o resultado coerente da soma dos dois anteriores. Isso reflete a estrutura coerente necessária para transmitir informações complexas na linguagem natural.

```
# Exemplo função fibonacci
def fibonacci(n):
if n <= 1:
return n
else:
return fibonacci(n - 1) + fibonacci(n - 2)
# Gerando a sequência de Fibonacci
termos = 10
sequencia = [fibonacci(i) for i in range(termos)]
print("Sequência de Fibonacci:", sequencia)
```

Ao destacar a coerência lógica presente nos exemplos de código Python, reforça-se a relevância do pensamento computacional no desenvolvimento da coerência tanto na programação quanto na linguagem natural. Essa conexão ilustra como a prática do pensamento computacional pode fortalecer a compreensão e a aplicação dos elementos de coesão e coerência na comunicação humana.

6 PROPOSTA DE UMA SEQUÊNCIA DIDÁTICA COM O USO DO SCRATCH

A sinergia entre linguagem natural e pensamento computacional, como discutido anteriormente, mostra grande promessa no desenvolvimento das funções psicológicas superiores. A linguagem natural, utilizada para a expressão e a comunicação humanas, e o pensamento computacional, com suas habilidades de decomposição, abstração, reconhecimento de padrões e algoritmos, têm o potencial de se reforçarem um ao outro. Isso, por sua vez, amplifica a criatividade, a resolução de problemas e a compreensão de conceitos complexos. Propõe-se, nesse contexto, uma sequência didática centrada na remidiação, utilizando-a como eixo para transformar narrativas construídas em linguagem natural em narrativas digitais interativas, empregando estratégias do pensamento computacional e a plataforma Scratch.

A remidiação, conforme proposta por Bolter e Grusin (2000), refere-se ao processo pelo qual novas mídias incorporam e reconfiguram as características de mídias anteriores. Trata-se de um conceito que sublinha a interação contínua entre diferentes formas de comunicação e suas influências mútuas, resultando em novas formas de expressão. Na perspectiva educacional, a remidiação pode ser vista como uma oportunidade de integrar a linguagem natural e o pensamento computacional, ampliando as possibilidades pedagógicas.

6.1 LINGUAGEM VISUAL DE PROGRAMAÇÃO SCRATCH

O Scratch é uma plataforma de programação visual desenvolvida pelo MIT (Massachusetts Institute of Technology) para ensinar conceitos de programação de maneira acessível e divertida. Ele foi projetado para ser usado por crianças, jovens e iniciantes em programação, oferecendo uma interface intuitiva baseada em blocos de código o que o torna mais atrativo em relação às linguagens tradicionais de programação.

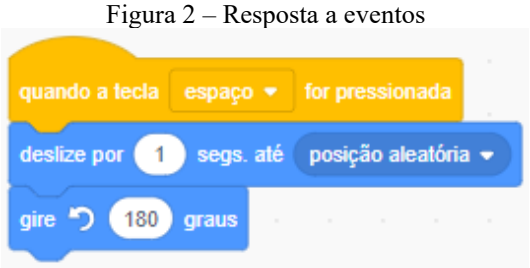
O Scratch permite que os usuários criem projetos interativos, como animações, jogos, histórias e muito mais, sem a necessidade de escrever código complexo. Sua interface gráfica utiliza blocos de comandos coloridos que se encaixam como peças de quebra-cabeça, facilitando a criação de sequências de ações. Os blocos de comandos representam diferentes operações, como movimento, aparência, som, controles e variáveis. Por exemplo, há blocos para mover personagens, reproduzir sons, alterar cores e controlar o fluxo do programa.

Figura 1 – Blocos para mover e alterar cor.

Fonte - Própria

Além disso, o Scratch permite criar atores e fundos para compor cenas. Os atores podem ser personalizados com imagens, sons e animações próprias. A tela do Scratch é dividida em duas áreas principais: a área de programação, onde os blocos são arrastados para criar scripts, e o palco, onde os projetos são executados. Os projetos no Scratch podem responder a eventos como cliques e teclas pressionadas. Isso permite que os programas reajam às ações do usuário.

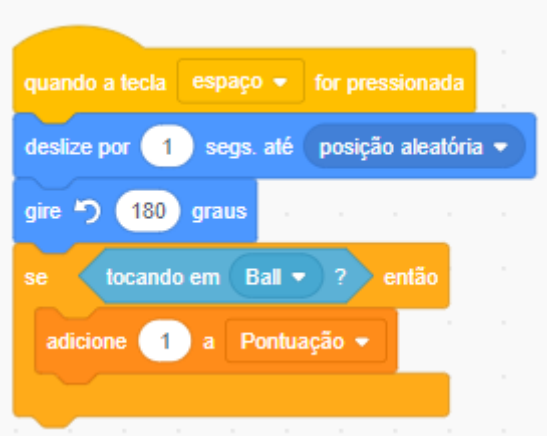
Figura 2 – Resposta a eventos



Fonte – Própria

O Scratch também permite o uso de variáveis para armazenar informações e controlar o comportamento dos programas. No exemplo abaixo, Pontuação é a variável que é modificada quando o personagem tocar na bola.

Figura 3 – Modificação de variável



Fonte – Própria

Enfim, o Scratch oferece uma série de recursos visuais e interativos que ajudam os usuários a compreenderem os princípios da programação de maneira prática e envolvente, promovendo a criatividade e o pensamento lógico.

6.2 PROPOSTA DE SEQUÊNCIA DIDÁTICA

Essa proposta de sequência didática visa amplificar os benefícios da sinergia entre o pensamento computacional, linguagem natural e as funções superiores psicológicas. Considerando as características do Scratch, é natural antecipar que, durante o processo de remidiação, a linguagem, inicialmente redigida em conformidade com as normas-padrão, passará por transformações substanciais. Isso resultará na adoção de uma variante linguística mais adequada ao novo ambiente digital da narrativa, promovendo, desse modo, o conceito de pedagogia da variação linguística, conforme destacado por Faraco (2015). Isso, por sua vez, sustenta a importância de uma abordagem pedagógica da língua portuguesa que esteja plenamente ciente do contexto comunicativo sociolinguístico. Nessa perspectiva, propõe-se a seguinte sequência didática:

1. Introdução ao conceito de remidiação e de pensamento computacional: Apresentar aos alunos o conceito de remidiação, destacando sua relação com a evolução das mídias e a interação entre a linguagem natural e o pensamento computacional. Introduzir os princípios fundamentais do pensamento computacional, tais como: decomposição, abstração, reconhecimento de padrões e algoritmos.
2. Análise de narrativas em linguagem natural: Incentivar os alunos a explorarem narrativas tradicionais e/ou autorais, identificando elementos narrativos, personagens, enredo e

postos-chave. Analisar as estruturas e técnicas de contação de histórias, levando em consideração os aspectos psicolinguísticos e emocionais dos personagens. Introduzir o processo de roteirização como parte integrante da transposição do texto em linguagem natural para uma narrativa digital interativa, com o uso do Scratch.

3. Roteirização e transposição para narrativas digitais interativas: Explicar a importância da roteirização na transformação das narrativas em linguagem natural para narrativas digitais interativas no ambiente Scratch. Apresentar o software Scratch como uma ferramenta para a construção de narrativas digitais interativas. Auxiliar os alunos na tradução dos elementos das narrativas linguísticas para a linguagem computacional, adaptando e recriando a história de forma interativa.
4. Desenvolvimento das narrativas interativas: Orientar os alunos na criação de seus próprios projetos no Scratch, integrando elementos da narrativa linguística adaptada. Estimular ativamente a aplicação do pensamento computacional, para solucionar desafios e criar interatividade na narrativa, levando em conta o roteiro previamente desenvolvido.

Considerando que a sequência didática proposta integra três componentes essenciais: texto em linguagem natural, processo de remidiação e narrativa digital interativa, e que possui como objetivo principal o desenvolvimento cognitivo dos alunos, serão apresentados, a seguir, alguns critérios para a avaliação final:

1. Fidelidade à narrativa original: Avaliar a fidelidade na transposição dos elementos fundamentais da narrativa em linguagem natural para o formato digital, assegurando a coesão da história durante a adaptação.
2. Criatividade na adaptação e na inovação: Avaliar a criatividade na reinterpretação e na adaptação da narrativa para uma forma interativa, levando em conta as inovações e os recursos utilizados para tornar a narrativa digital única e envolvente.
3. Eficiência na utilização do software Scratch: Avaliar a habilidade dos alunos em utilizarem, de forma eficiente, as funcionalidades do Scratch, para criar interatividade e cativar o público-alvo, garantindo uma experiência fluida e atrativa.
4. Aplicação proficiente do pensamento computacional: Avaliar a aplicação sólida dos princípios do pensamento computacional, tais como: decomposição, abstração, reconhecimento de padrões e algoritmos, durante o desenvolvimento da narrativa digital interativa, demonstrando compreensão e habilidade na aplicação desses princípios.

A integração da remidiação com a abordagem pedagógica proposta oferece uma oportunidade valiosa para o enriquecimento da experiência educacional, unindo a expressão humana por meio da linguagem natural à potência criativa do pensamento computacional. Essa sequência didática fornece uma estrutura sólida para a exploração desses conceitos, incentivando a criação de narrativas digitais

interativas que não apenas ampliam a compreensão e aplicação do pensamento computacional, mas também enriquecem a expressão criativa dos alunos. Esse processo, por conseguinte, promove o desenvolvimento cognitivo dos estudantes de maneira abrangente.

Dito isso, é fundamental ressaltar como essa proposta de sequência didática, com o uso do Scratch, vai além da integração entre linguagem natural e pensamento computacional. Ela se torna um pilar fundamental no fortalecimento das capacidades dos alunos em compreender e aplicar habilidades essenciais para a resolução de problemas e expressão criativa. Ao adentrar no universo do Scratch, os alunos não estão apenas aprendendo a usar uma ferramenta; estão imersos em uma linguagem visual de programação que permite a construção de narrativas interativas. Essa abordagem prática oferece a eles uma oportunidade significativa de:

1. Pensar de forma algorítmica: A estrutura de blocos no Scratch ensina os alunos a pensar em sequências lógicas de ações. Ao organizar comandos de maneira lógica para criar interações desejadas, eles desenvolvem a habilidade de decompor problemas em passos menores e solucionáveis.
2. Abstrair e reconhecer padrões: Enquanto criam projetos no Scratch, os alunos identificam padrões visuais e funcionais nos blocos de programação. Isso ajuda na compreensão de conceitos como repetição, condicionais e loops, capacitando-os a abstrair esses padrões para aplicação em diferentes contextos.
3. Resolver problemas de maneira criativa: A plataforma Scratch estimula a criatividade ao permitir que os alunos expressem suas ideias de forma interativa. Eles são desafiados a encontrar soluções originais para implementar funcionalidades, promovendo a criatividade na resolução de problemas.
4. Aprimorar habilidades de comunicação e expressão: Ao transformar narrativas em projetos interativos, os alunos são incentivados a expressar suas ideias de forma clara e coesa, aplicando conceitos de linguagem natural na construção de histórias digitais.

Dessa forma, essa abordagem não se limita ao aprendizado de programação. Ela oferece uma oportunidade única para os alunos desenvolverem habilidades fundamentais para o pensamento computacional, ao mesmo tempo em que promove a expressão criativa e a resolução de problemas de maneira inovadora e significativa. Essa integração entre remidiação, linguagem natural e pensamento computacional não só enriquece a experiência educacional, mas também prepara os alunos para enfrentarem os desafios e explorarem as oportunidades do mundo moderno.

7 CONCLUSÃO: UMA SINERGIA E UMA SINFONIA

Em nossa imersão pelas profundezas da sinergia entre linguagem natural e pensamento computacional neste capítulo, exploramos os intrincados caminhos que entrelaçam a riqueza

expressiva da comunicação humana com a lógica estruturada do pensamento computacional. A proposta de uma sequência didática envolvendo a remidiação e a exploração do Scratch representa mais do que um simples mergulho na programação visual; é uma jornada profunda no potencial educacional que expande os horizontes convencionais da sala de aula.

Iniciamos nossa jornada com um convite a uma metamorfose linguística. Ao adentrar o universo do pensamento computacional, apresentamos os pilares fundamentais: decomposição, abstração, reconhecimento de padrões e algoritmos. Esses princípios não são apenas ferramentas para resolver desafios lógicos; são instrumentalidades que aprimoram a capacidade de análise, criação e inovação. E é na intersecção desses princípios com a linguagem natural que residem as bases de nossa proposta educacional.

Nosso ponto de partida para a sequência didática proposta foi a análise minuciosa das narrativas em linguagem natural, convidando os alunos a explorar as complexidades da expressão humana. Esse mergulho permite identificar elementos narrativos, compreender estruturas emocionais e psicolinguísticas, sendo os primeiros acordes dessa sinfonia educacional. Essa análise minuciosa estabelece as bases para o próximo passo: a jornada da roteirização, entrelaçada com o potencial do Scratch. Aqui, os alunos não apenas traduzem, mas reinventam as narrativas, adaptando-as ao cenário digital emergente, nutrindo assim a riqueza da variação linguística sugerida por Faraco (2015). Essa transição natural permite que o universo da linguagem natural se funda harmoniosamente com as possibilidades inovadoras e criativas proporcionadas pelo ambiente digital. O Scratch, nesse panorama, delineia um caminho onde a criatividade e o pensamento computacional convergem harmoniosamente. Através de seus blocos coloridos, os alunos transformam ideias em interações visuais, aplicando os princípios do pensamento computacional de maneira prática e envolvente. Cada linha de código no Scratch é um verso, e cada projeto é uma obra única, uma expressão digital de narrativas remidiadas.

Ao explorarmos o desenvolvimento das narrativas interativas, não apenas incentivamos a aplicação do pensamento computacional, mas também o florescer da criatividade. Os alunos tornam-se protagonistas, criadores de suas próprias histórias digitais, onde cada desafio superado é uma nota harmoniosa na melodia da aprendizagem.

A avaliação final não é um julgamento, mas um convite à reflexão. Fidelidade à narrativa original, criatividade na adaptação, eficiência no uso do Scratch e aplicação proficiente do pensamento computacional são critérios que não buscam apenas medir, mas iluminar o progresso, reconhecendo não apenas o resultado, mas o percurso trilhado pelos aprendizes.

Neste mundo onde a interação entre linguagem natural e pensamento computacional é essencial, nossa proposta pedagógica se mostra como uma trilha sonora educacional única. Convida educadores a se tornarem maestros, alunos a se tornarem compositores, transformando a sala de aula em um palco de expressão, aprendizado e descoberta.

Este capítulo não é apenas um ponto de chegada, mas um ponto de partida para uma abordagem educacional que transcende paradigmas. A sinergia entre linguagem natural, pensamento computacional e educação é a sinfonia que ressoa na construção de cidadãos do futuro, criativos, inovadores e dotados da habilidade única de traduzir pensamentos em linguagem digital.

Em última análise, este capítulo se apresenta não apenas como uma contribuição à pesquisa, mas como uma ferramenta prática e valiosa para educadores que buscam enriquecer suas práticas pedagógicas. A proposta não é apenas teoria; é um convite à transformação, à criação de experiências educacionais que ecoam além dos confins das salas de aula, moldando mentes para os desafios e oportunidades do século XXI. Que esta sinfonia educacional inspire, encante e guie os passos de todos os envolvidos na arte de educar.

REFERÊNCIAS

BEAUGRANDE, Robert-Alain de; DRESSLER, Wolfgang Ulrich W. Introduction to text linguistics. Tübingen, Germany: Max Niemeyer, 1981.

BOLTER, Jay David; GRUSIN, Richard. Remediation: Understanding new media. Cambridge, Massachusetts: MIT Press, 2000.

FARACO, Carlos Alberto. Norma culta brasileira: construção e ensino. *In*: ZILLES, Ana Maria Stahl; FARACO, Carlos Alberto (org.). Pedagogia da variação linguística: língua diversidade e ensino. São Paulo: Parábola Editorial, 2015. p. 19-30.

KOCH, Ingedore. Introdução à linguística textual. São Paulo: Contexto, 2020.

OPERATIONAL Definition of Computational Thinking – for K-12 Education. [*S. l.*]: NSF; ISTE; CSTA, 2011. Disponível em: https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf. Acesso em: 28 set. 2023.

VIGOTSKY, Lev Semenovich. Pensamento e Linguagem. São Paulo, SP: Ícone, 2001. *E-Book*. Edição eletrônica: Ed. Ridendo Castigat Mores. Disponível em: www.jahr.org. Acesso em: 26 out. 2023.

VIGOTSKY, Lev Semenovich; LURIA, Alexander Romanovich; LEONTIEV, Alex N. Linguagem, desenvolvimento e aprendizagem. Tradução: Maria da Pena Villalobos. 11. ed. São Paulo, SP: Ícone, 2010. (Coleção Educação Crítica).