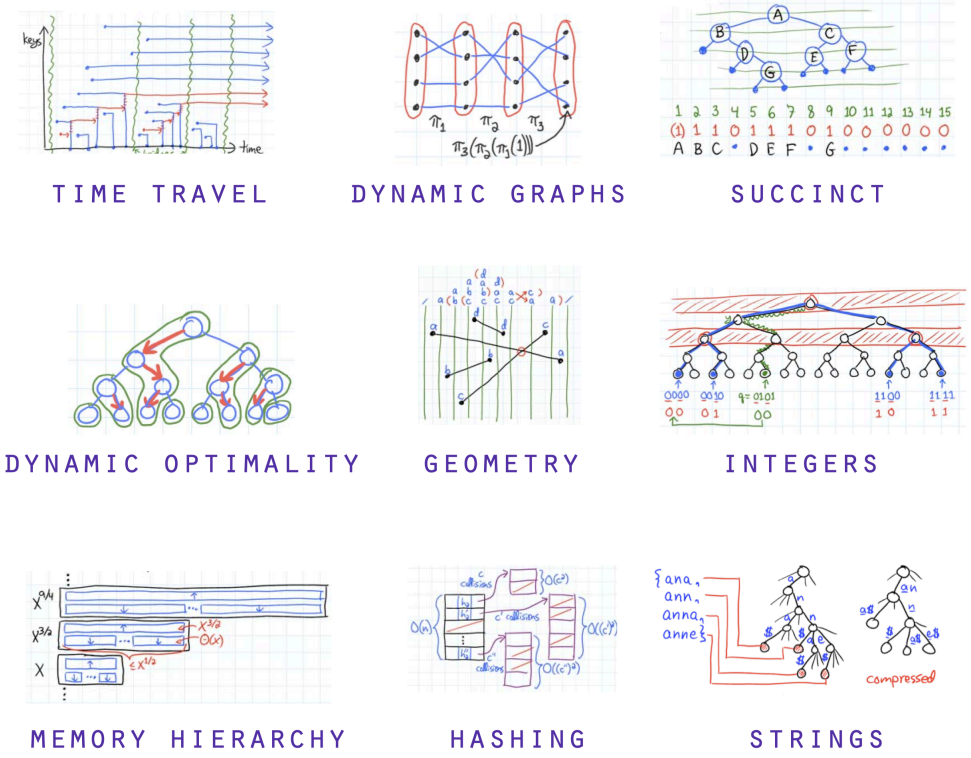
Estrutura de Dados

Modularização em Python

Prof. Saulo Oliveira

Análise e Des. de Sistemas



Agenda

- Funções
- Recursão
- Bibliotecas
- Módulos

Funções

> ⚠ Um problema complexo pode ser simplificado quando dividido em vários problemas menores que são mais fáceis de resolver.

Decomposição:

- Redução de complexidade;
- Permite focalizar a atenção em um problema pequeno de cada vez;
- Produz melhor compreensão do todo.

Analogia com o corpo humano:

- Corpo humano 🧍 ≺ Sistemas 👱 ≺ Órgãos 🫀 ≺ Células ⚛ ≺ Móleculas 🧬.
- Módulos ≺ Bibliotecas ≺ Funções ≺ Funções ≺ Variáveis.

Funções

As funções são blocos de instruções que realizam tarefas específicas;
O código de uma função é carregado uma vez e pode ser executado quantas vezes forem necessárias.

- Os programas em geral são executados linearmente, uma linha após a outra, até o fim;
- As funções permitem a realização de desvios na execução dos programas;
- Desvios são efetuados quando uma função é chamada pelo programa principal.

Funções

Vejamos o exemplo da função `len` no Python.

Definição: Retorna o comprimento (o número de itens) de um objeto. O argumento pode ser uma sequência (como uma `string`, `bytes`, `tuple`, `list` ou `range`) ou uma coleção (como um `dict`, `set` ou `frozenset`).

```python
nome = input('Digite seu nome')
l = len(nome)
print(l)
```

```python
# bultins.py -- fake
def len(s: Sized) -> int:
    contador = 0

    for _ in s:
        contador += 1

    return contador
```

Anatomia de uma função (1)

- Na **definição** de funções, as variáveis recebem um nome e são chamadas de **parâmetros**;
- Funções processam algo e devolvem/retornam valores após sua execução. Para isto, utiliza-se o comando `return` e valores são retornados.
- Para capturar os valores retornados, as funções devem aparecer do lado direito de uma expressão de atribuição;
- Usamos *dicas* para os sinalizarmos os tipos de valore. Usamos duas sintaxes, a saber, `: int` para variáveis e `-> int` para funções.

```python
#somatorio.py

def somatorio(n: int) -> int:
    total = 0

    for i in range(n + 1):
        total += i

    return total

s1 = somatorio(10)
s2 = somatorio(4)
s3 = somatorio(-3)

print(f'A soma até 10 é {s1}.')
print(f'A soma até 4  é {s2}.')
print(f'A soma até -3 é {s3}.')
```

Escopo de variáveis

Escopo: refere-se à abrangência em que uma variável estará disponível no seu programa. Na maioria das linguagens de programação há dois escopos, a saber, o `global` e o `local`.

- As variáveis que são declaradas com o **escopo global** estão disponíveis em qualquer região de seu programa, independentemente do tamanho que seu programa possua;
- Já as variáveis de **escopo local** estão disponíveis, apenas, na região em que foram declaradas.

Por exemplo, uma variável que foi definida dentro de uma função existe apenas dentro daquela função. Após a execução e o encerramento de uma função, essa variável não mais existirá e, se o seu valor (conteúdo) não for armazenado em uma variável global, ele será descartado.

Passagem por valor e passagem por referência

Fonte: https://devblog.drall.com.br/excelente-imagem-animada-gif-para-ensinarrepresentar-passagem-de-parametros-por-valorcopia-e-por-referencia.

Modularização

Typing

> ⚠ **Python** não força anotações de tipos de variáveis e funções em tempo de execução. No entanto, ferramentas de terceiros como verificadores de tipo, IDEs, linters, etc, as usam.

Considere a função abaixo:

```
def echo(text: str, n: int) -> str:
    return text * n
```

- A função `echo` recebe dois argumentos que se espera ser do tipo `str` e do tipo `int`, respectivamente;
- Espera-se que a função retorne uma `str`, conforme indicado pela *dica* `-> str`.
- O módulo `typing` fornece *dicas de tipo* mais avançadas;

TAD de um cubo 🧊

Desenvolva um TAD que represente um cubo.

Inclua as funções de inicialização necessárias e as operações que retornem o tamanhos de cada lado, a sua área e o seu volume.

```python
# cubo.py
class Cube:

    def __init__(self, side: float) -> None:
        raise NotImplemented

    @property
    def side(self) -> float:
        raise NotImplemented

    def area(self) -> float:
        raise NotImplemented

    def volume(self) -> float:
        raise NotImplemented
```

Referências

- Python Software Foundation. **typing — Support for type hints**. 2024. Disponível em: https://docs.python.org/3/library/typing.html