# Typing

**Python é conhecido por ser uma linguagem de tipagem dinâmica, ou seja uma variável poder mudar seu tipo de dado durante o tempo de execução conforme a necessidade do programa,** porém em alguns momento temos a necessidade de que certos dados sigam certos padrões, principalmente quando tratamos de lista, dicionários, tuplas e funções

- Assim surge a necessidade de por exemplo um **dicionário seguir certos padrões** durante seu desenvolvimento
- **Quando vamos construir uma função criamos uma lógica onde a mesma receba certos valores** (Por exemplo uma função de soma, onde esperasse que seus valores sejam inteiros (int)) porém quando não fazemos essa tipagem pode ser inserido uma string que o programa não aponta um erro, somente quando o mesmo é executado que o problema é identificado

Em resumo, todos esses casos precisamos da ajuda do Typing e suas funções que tornam essas estruturas estáticas, recebendo só um padrão de valor

---

## Tipagem de uma Função

Quando fazemos isso estamos dando dicas ao código de como deve ser comportar os parâmetros da função (Informando na hora de passar o parâmetro qual o tipo esperado para a função funcionar)

```
def funcao1 (valor1: float, valor2: float) -> float
    return (valor1+valor2)/2
```

Nesse exemplo fazemos a soma de dois valores e dividimos por dois, nesse caso é esperado dados do tipo int ou float, tendo em vista que uma divisão só pode ser realizado com valores numéricos

Porém se for passado um valor str o programa não aponta erro, somente no momento da sua execução, para evitar esse problema usamos as dicas

Nesse caso é esperado que o valores são do tipo float, além da função fazer o retorno do tipo float, facilitando na hora de definir seu valores além de documentar de certa forma a função e fazer uma prevenção contra erros

```
def funcao2 (valor1: float, valor2: flo
    return (valor1+valor2)/2
```

👉 Uma das vantagens desse método é trazer um certo questionamento por parte dos programadores na hora de fazer uma função, pois é necessário pensar no tipo que cada variável terá bem como o formato de retorno de uma função

Juntamente com o Mypy o editor verificar no momento de criar uma nova instância para a função apontando possíveis erros

`__annotations__` → Comando que verifica as anotações de uma função, ou seja quais as dicas/tipagem de uma função, bem como o tipo de variáveis presentes dentro de uma função

- `Reveal_Type()` → Retorna o tipo do objeto passado como parâmetro da função
- `Reveal_Locals()` → Retorno o tipo de todas as variáveis locais do programa, facilitando assim a depuração do mesmo

---

## Tipagem de Lista

Dentro desse módulo, também podemos fazer anotações dentro de lista, que possibilitam que a mesma receba somente o tipo de valor informado

👉 Sempre lembrar de importa os módulos para isso

```
from typing import List
```

E nesse caso precisamos o módulo list

Para poder usar esse módulo seguimos a seguinte sintaxe

`minha_lista: List[int] = []`

Onde precisamos informar que a lista terá como referência ao módulo Typing

---

## Tipagem de Dicionário

Um dos pontos desse módulo está no fato de conseguir criar um novo estilo de dicionário, com capacidade de validação pelo próprio editor de texto, apontando possíveis erros (Como nome de chaves ou tipo de valor errado)

- Como mencionado esse módulo de tipagem permite criar um molde para dicionário com chaves e valores padrões através de um conceito de classe

```
from typing import TypedDict

class Pessoa (TypedDict):
    nome: str
    idade: int
```

Nesse exemplo criamos um padrão de dicionário que só será valido se for informado uma chave com nome e outra idade, além do tipo dos valores serem compatíveis com o tipo informado na sua declaração (str e int), caso o contrário será apontado erro pelo Mypy

No momento de criar um nova instância com base nesse dicionário usamos a seguinte sintaxe:

```
Pessoa1: Pessoa = {'nome': 'João', 'idade': 15}
```

Nesse exemplo criamos uma variável que segue como modelo oque foi definido na classe Pessoa, garantindo validação de chaves e valores

> 👉 No Python a definição do tipo da variável vem após o uso do `:` nesse caso foi tipado com o molde Pessoa

Após isso foi definido os valores

---

## Tipagem de Tupla

Dentro desse módulo, também podemos informa o valor padrão recebido dentro de uma tupla

```
x1: tuple[int, str, int] = (1, 'L', 30)
```

Nesse exemplo foi informado que uma tupla só será completa caso seja satisfeito o padrão de um número inteiro, seguido por uma string e por fim outro número inteiro, caso o contrário o verificador de tipos era apontar um erro

---

## NewType

Esse comando atua como uma barreira para o código, pois você cria um novo tipo de variável, onde por exemplo, você pode **criar esse tipo como segurança para um processo dentro do código.** Um valor só pode ser tratado como `UserId` se for explicitamente convertido usando `UserId(valor)`. Caso contrário, mesmo que seja um número inteiro, ele será tratado apenas como `int`, e o Mypy impedirá seu uso como `UserId` em funções que exigem esse tipo.  Ou qualquer outras aplicações, porém que você precise de um "tipo específico" ou mais conhecido como subtipo (Nada mais é doque um tipo comum com uma FLAG)

```
import typing

UserId = NewType('UserId', int)
```

O primeiro argumento é justamente o nome da FLAG - No caso o nome que vai aparecer no momento da depuração do código, mais usado pelos verificador de código, o próximo argumento é o tipo desse novo tipo

---

## Apelido

Forma simplificado de chamar um tipo específico de dado Ex:

Temo uma tupla em formato de coordenadas e você usa esse padrão em vários momentos, então você renomeia apelidando por exemplo para coordenada e atribui um nova tupla com esse tipo como se fosse um outro tipo padrão da linguagem

```
Coordenada = tuple[int, str, int]
```

Nesse momento criamos um apelido para esse padrão de tupla, e quando necessário, atribuímos a uma outra variável esse tipo, denominando que ela siga o padrão

```
c1: Coordenada = (24, 'SW' 40)
```