

1



2



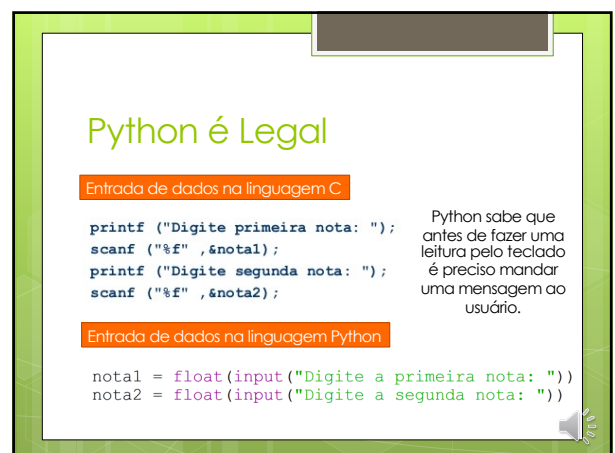
3



4



5



6

## Processamento




Chip Único:  
Unidade de Controle  
Unidade Lógica e Aritmética

Memória

7

## Memória



**Variáveis:** Espaço na memória onde se armazenam dados de um determinado tipo.

**name = conteúdo**  
Lê-se:  
[variável] recebe [conteúdo].


**fator = 5**  
Lê-se:  
fator recebe cinco.

Endereço	Dado
...	?
0065	?
0066	?
0067	5
0068	?
0069	?
0070	?
...	?

Fator

8

## Unidade Lógica e Aritmética



**Operadores Aritméticos**

Operador	Função	Exemplo	Resultado
=	Atribuição (recebe)	$n = 10$	10
+	Adição	$n = 1 + 2$	3
-	Subtração	$n = 1 - 2$	-1
*	Multiplicação	$n = 2 * 3$	6
/	Divisão (Real)	$n = 5 / 2$	2.5
//	Divisão (Inteira)	$n = 5 // 2$	2
**	Exponenciação	$n = 5 ** 2$	25
%	Módulo (Resto da Divisão)	$n = 5 \% 2$	1

9

## Unidade Lógica e Aritmética



**Operadores Lógicos**

Operador	Função
and	E lógico
or	OU lógico
not	NÃO lógico

E (and)	V	F	OU (or)	V	F
V	V	F	V	V	V
F	F	F	F	V	F

NÃO (not)	V	F
-	F	V

10

## Unidade de Controle

Define qual comando será executado.

**Estrutura Sequencial:** Todos os Comandos são Executados em Ordem:

```

graph LR
    Inicio([Inicio]) --> C1[C1]
    C1 --> C2[C2]
    C2 --> Dots[...]
    Dots --> Cn[Cn]
    Cn --> Fim([Fim])
  
```

11

## Unidade de Controle

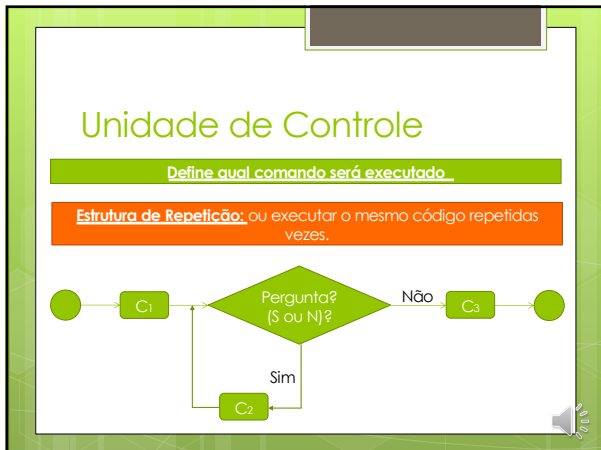
Define qual comando será executado.

**Estrutura Condicional:** em algum momento precisamos decidir entre um caminho ou outro

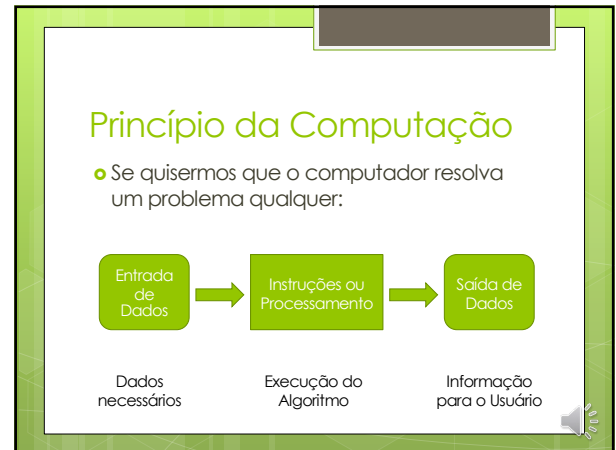
```

graph LR
    Start(( )) --> C1[C1]
    C1 --> Dec{Pergunta? (S ou N)?}
    Dec -- Sim --> C2[C2]
    Dec -- Não --> C3[C3]
    C2 --> End(( ))
    C3 --> End
  
```

12



13



14

## Princípio da Computação

- Se quisermos que o computador resolva um problema qualquer:

```

# Entrada de dados
a = int(input('Valor de A: '))
b = int(input('Valor de B: '))

# Processamento
soma = a + b

# Saída de Dados
print(f'A soma de {a} com {b} é {soma}')
  
```

15

## Programas São Grandes

- Crie subprogramas (funções)
  - com ou sem Entrada de Dados
    - O processamento que será feito precisa de alguma informação?
  - com ou sem Retorno de Dados
    - Foi realizado algum cálculo cujo resultado precisa ser devolvido?

16

## Funções

### Sem Entrada de Dados

```
def diga_ola():
    print("Olá!")
```

→ `diga_ola()`

→ Funções são chamadas para execução pelo seu nome e lista de argumentos necessários.

### Com Entrada de Dados (Argumentos)

```
def diga_nome(nome, idade):
    print(f"Olá! {nome} tem {idade} anos")
```

→ `diga_nome("João", 25)`

17

## Funções

### Sem Retorno de Dados

```
def diga_ola():
    print("Olá, mundo!")
```

`diga_ola()`

### Com Retorno de Dados

```
def dobro(n):
    return n * 2
```

`d = dobro(27)`  
`print(d)`

Funções com retorno efetuam um cálculo e retornam o valor. NÃO devem ter comandos de `input()` ou `print()`.

18

## Funções

### Com Retorno de Dados

```
def minutos(horas, minutos):  
    m = (horas * 60) + minutos  
    → return m
```

```
h = int(input("Horas: "))  
m = int(input("Minutos: "))  
m = minutos(h, m)  
print(f'Minutos: {m}')
```

Os dados de entrada são recebidos como parâmetros (separados por vírgula) e a saída é feita com **return**.

19

## Funções

### Com Retorno de Dados

```
def horas_minutos(minutos):  
    h = minutos // 60  
    m = minutos % 60  
    → return h, m  
  
m = int(input("Minutos: "))  
→ h, m = horas_minutos(m)  
print(f'{h}{m}min')
```

Também é possível retornar vários valores (separados por vírgula).

É preciso ter a mesma quantidade de variáveis para receber o retorno da função.

20

## Boa Prática em Python

Criar uma função **main()** que é a principal (onde o programa começa e termina).

```
def mensagem(msg):  
    print(msg)  
  
def main():  
    mensagem("Alô, Mundo!")  
  
if __name__ == '__main__':  
    main()
```

21

## Sobre o uso de FUNÇÃO

- Prefira sempre criar uma função para realizar tarefas específicas.
- No princípio, pode parecer um trabalho desnecessário mas acredite: a organização do código VALE A PENA.
- Não deve usar **input()** ou **print()** em funções que efetuam cálculos.
- Funções sem retorno podem usar o comando **print()**.

22

```
v = float(input("Valor R$: "))  
p = float(input("Desejado %: "))  
  
v = v * (p / 100)  
  
print(f"Desejado R$: {v:.2f}")
```

Iniciantes em programação acreditam que sem funções o código fica menor. É um **ERRO** pensar assim.



23

```
def percentual():  
    v = float(input("Valor R$: "))  
    p = float(input("Desejado %: "))  
  
    v = p * (v / 100)  
  
    print(f"Desejado R$: {v:.2f}")  
  
percentual()
```

Também é um **ERRO** criar uma função que faz a TUDO (entrada, processamento e saída). Cada função deve realizar uma tarefa específica.



24

```
def percentual(valor, porcentagem):
    return valor * (porcentagem / 100)

v = float(input("Valor R$: "))
p = float(input("Desejado %: "))

v = percentual(v, p)

print(f"Desejado R$: {v:.2f}")
```

Usando FUNÇÕES específicas  
você melhora a leitura e fica  
mais fácil para reutilizar o código  
depois de feito.



25

```
def percentual(valor, porcentagem):
    return valor * (porcentagem / 100)

def main():
    v = float(input("Valor R$: "))
    p = float(input("Desejado %: "))

    v = percentual(v, p)

    print(f"Desejado R$: {v:.2f}")

if __name__ == '__main__':
    main()
```

Melhor ainda é seguir as boas práticas  
de programação em Python e criar  
sempre uma função **main()**



26