Folha Resumo de Python

Tipos Base

inteiros, reais, lógicos, cadeias

```
  int 783    0    -192
float 9.23   0.0   -1.7e-6        (1.7e-6 = 10^-6)
 bool True   False
  str "Um\nDois"   'Pa\'mim'
```

- nova linha
- ' escaped
- multilinha: `"""X\tY\tZ 1\t2\t3"""` (tabulação)
- cadeia imutável, sequência ordenada de letras

Tipos Containers

- sequência ordenada, índices rápidos, valores repetíveis

```
  list [1,5,9]   ["x",11,8.9]   ["texto"]   []
 tuple (1,5,9)   11,"y",7.4     ("texto",)  ()
```

Expressão separada por virgulas

imutável → `str` como sequência ordenada de caracteres

- sem ordem anterior, chave única, índices rápidos ; chaves = tipos base ou tuplas

```
  dict {"chave":"valor"}                    {}
dicionário {1:"um",3:"três",2:"dois",3.14:"π"}
```

associações chave/valor

```
  set {"key1","key2"}   {1,9,3,0}   set()
```

Identificadores

para variáveis, funções, módulos, classes… nomes

`a..zA..Z_` seguidos de `a..zA..Z_0..9`

- acentos permitidos mas melhor evitar
- proibido usar palavras reservadas python
- distingue minúsculas/MAIÚSCULAS

☺ `a toto x7 y_max BigOne`
☹ ~~`8y and`~~

Atribuição de Variáveis

```
x = 1.2+8+sin(0)
```

valor ou expressão calculada

nome de variável (identificador)

```
y,z,r = 9.2,-7.6,"bad"
```

nome de variável | container com vários valores (aqui uma tupla)

`x+=3` somar

`x-=2` subtrair

`x=None` «indefinido» valor constante

Conversões

`tipo(`*expressão*`)`

`int("15")` podemos especificar a base no 2º parâmetro

`int(15.56)` trunca a parte decimal (`round(15.56)` para arredondar)

`float("-11.24e8")`

`str(78.3)` e a representação literal → `repr("Texto")`

ver o verso para descobrir como formatar cadeias

`bool` → use comparadores (com `==`, `!=`, `<`, `>`, …), resultado lógico

`list("abc")` → *use cada elemento de uma sequência* → `['a','b','c']`

`dict([(3,"três"),(1,"um")])` → `{1:'um',3:'três'}`

`set(["um","dois"])` → *use cada elemento de uma sequência* → `{'um','dois'}`

`":".join(['toto','12','pswd'])` → `'toto:12:pswd'`

`":"` unir cadeias; `['toto','12','pswd']` sequência de cadeias

`"palavras e espaços".split()` → `['palavras','e','espaços']`

`"1,4,8,2".split(",")` → `['1','4','8','2']`

`","` separar cadeias

Índices de sequências

para listas, tuplas, cadeias, …

`len(lst)` → `6`

índices negativos	-6	-5	-4	-3	-2	-1
índices positivos	0	1	2	3	4	5
`lst=`	`[11,`	`67,`	`"abc",`	`3.14,`	`42,`	`1968]`
corte positivo	0	1	2	3	4	5
corte negativo	-6	-5	-4	-3	-2	-1

acesso individual aos valores `[`*índice*`]`

`lst[1]`→`67` `lst[0]`→`11` *primeiro valor*

`lst[-2]`→`42` `lst[-1]`→`1968` *último valor*

acesso a sub-sequências via `[`*inicio corte*`:`*fim corte*`:`*passos*`]`

```
lst[:-1]→[11,67,"abc",3.14,42]
lst[1:-1]→[67,"abc",3.14,42]
lst[::2]→[11,"abc",42]
lst[:]→[11,67,"abc",3.14,42,1968]
lst[1:3]→[67,"abc"]
lst[-3:-1]→[3.14,42]
lst[:3]→[11,67,"abc"]
lst[4:]→[42,1968]
```

Omitindo o parâmetro de corte → de principio / até o fim.

Em sequências mutáveis, pode-se eliminar elementos com `del lst[3:5]` *modificar com designação* `lst[1:4]=['hop',9]`

Lógica Booleana

Comparadores: `<` `>` `<=` `>=` `==` `!=` (≤ ≥ = ≠)

`a and b` 'e' lógico — *ambos simultaneamente*

`a or b` 'ou' lógico — *um, outro, ou ambos*

`not a` 'não' lógico

`True` valor constante verdadeiro

`False` valor constante falso

Bloco de Sentenças

sentença mãe :
→ *bloco de sentenças 1…*
⋮
sentença mãe :
→ *bloco de sentenças 2…* □

¡sangría!

sentença depois o bloco 1

Sentenças Condicionais

bloco de sentenças que só será executado se a condição é verdadeira

`if` *expressão lógica* `:`
→ *bloco de sentencias*

pode ter vários *elif*, *elif*... e só um *else* ao final, exemplo:

```
if x==42:
    # só se a expressão lógica x==42 é verdadeira
    print("Realmente verdadeira")
elif x>0:
    # se não, se a expressão lógica x>0 é verdadeira
    print("Somos positivos")
elif estamosProntos:
    # se não, se a variável lógica é verdadeira
    print("Sim, estamos prontos")
else:
    # nos demais casos
    print("Todo o anterior não foi")
```

Matemáticas

números reais… valores aproximados! — *ângulos em radianos*

Operadores: `+` `-` `*` `/` `//` `%` `**` (× ÷ ÷ inteiros, resto de ÷, a^b)

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
acos(0.5)→1.0471…
sqrt(81)→9.0   √
log(e**2)→2.0   etc. (cf doc)
```

Sentença Loop Condicional

bloco de sentencias repetido enquanto a condição é certa

```
while expressão lógica:
    bloco de sentencias
```

```
s = 0
i = 1
```
*inicializações **antes** do laço*

Condição com pelo menos um valor variável (aqui `i`)

```
while i <= 100:
    # sentencias executam-se enquanto i ≤ 100
    s = s + i**2
    i = i + 1
```
alteramos *o valor condicional*

$$s = \sum_{i=1}^{i=100} i^2$$

```
print("suma:",s)
```
resultado depois do laço

Controle de Loop

```
break
```
sair imediatamente

```
continue
```
próxima iteração

Sentença Loop Iterador

bloco de sentencias executadas para cada item de um contenedor ou iterador

```
for variável in sequência:
    bloco de sentenças
```

verifique os **valores** da sequência

```
s = "um texto"
cnt = 0
```
*inicializamos **antes** do laço*

variável do laço, valor manejado pela sentencia `for`

```
for c in s:
    if c == "t":
        cnt = cnt + 1
print("encontramos",cnt,"'t'")
```
Conte a quantidade de letras `t` na cadeia

loop de um dict/set = loop a sequência de chaves
use cortes para verificar uma subsequência

Verifique os **índices** de uma sequência
- modificar o item no índice
- acessar itens em volta do índice (antes/depois)

```
lst = [11,18,9,12,23,4,17]
perdidos = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        perdidos.append(val)
        lst[idx] = 15
print("modif:",lst,"-perd:",perdidos)
```
Limita os valores maiores a 15, guarda os valores perdidos.

Verificar simultaneamente os **índices** e **valores** de uma sequência:

```
for idx,val in enumerate(lst):
```

Entrada / Saída

```
print("v=",3,"cm :",x,",",y+4)
```
itens a escrever: valores literais, variáveis, expressões

parâmetros de `print`:
- `sep=" "` (separador de itens, por padrão espaço)
- `end="\n"` (caractere final, por padrão nova linha)
- `file=f` (escrever arquivo, por padrão saída estandard)

```
s = input("Instruções:")
```
`input` sempre retorna uma **cadeia**, converter ao tipo requerido (revisar *Conversões* ao verso).

Operações sobre Containers

`len(c)` → contagem de itens

`min(c)` `max(c)` `sum(c)`

*Nota: Para dicionários e conjuntos, estas operações usam as **chaves**.*

`sorted(c)` → *cópia* ordenada

`valor in c` → lógico, operador de presença `in` (ausência `not in`)

`enumerate(c)` → *iterador* sobre (índice, valor)

*Especial para **containers de sequências** (listas, tuplas, cadeias) :*

`reversed(c)` → *iterador reverso* `c*5` → duplicados `c+c2` → concatenar

`c.index(val)` → posição `c.count(val)` → conta ocorrências

Operações sobre Listas

modificar lista original

`lst.append(item)`	adicionar item ao final
`lst.extend(seq)`	adicionar sequência de itens ao final
`lst.insert(idx,val)`	adicionar item ao índice
`lst.remove(val)`	elimina primeiro item com valor
`lst.pop(idx)`	elimina item do índice e retorna seu valor
`lst.sort()` `lst.reverse()`	ordena / reverte a lista original

Operações em Dicionários

`d[chave]=valor` `d.clear()`

`d[chave]`→*valor* `del d[chave]`

`d.update(d2)` *atualiza/adiciona associações*

`d.keys()` `d.values()` `d.items()` *ver chaves, valores e associações*

`d.pop(chave)`

Operações em Conjuntos

Operadores:
- `|` → união (barra vertical)
- `&` → interseção
- `-` `^` → diferença/diferença simétrica
- `<` `<=` `>` `>=` → relações de inclusão

`s.update(s2)` `s.add(chave)`

`s.remove(chave)`

`s.discard(chave)`

Gerador de Sequências de Inteiros

uso frequente em loop iterativos `for`

por padrão 0 — não inclusivo

```
range([inicio,]fim [,passo])
range(5)      → 0 1 2 3 4
range(3,8)    → 3 4 5 6 7
range(2,12,3) → 2 5 8 11
```
`range` retorna um « gerador », converter em lista para ver os valores, por exemplo:

```
print(list(range(4)))
```

Definir Funções

nome da função (identificador) — parâmetros nomeados

```
def nomefunc(p_x,p_y,p_z):
    """documentação"""
    # bloco de sentenças, calcula result., etc.
    return res
```
valor resultado. se não ha resultado, então retorna: `return None`

parâmetros e variáveis desse bloco Só existem *dentro* do bloco e *durante* a chamada à função (*"caixa preta"*)

Invocar Funciones

```
r = nomefunc(3,i+2,2*i)
```
um argumento por parâmetro

obter o valor de retorno (se necessário)

Arquivos

gravar dados no disco, reler os dados

```
f = open("doc.txt","w",encoding="utf8")
```
- **variável** para operações
- **nome** do arquivo (+caminho…) — consulte funções nos módulos `os` e `os.path`
- **modo** de abertura
  - `'r'` ler
  - `'w'` escrever
  - `'a'` adicionar…
- **codificação** de caracteres em arquivo de texto: `utf8` `ascii` `latin1` …

escritura

```
f.write("oi!")
```
*arquivo de texto → lê / escreve só **textos**, converte de/para tipo requerido.*

leitura

Cadeia vazia se fim de arquivo

```
s = f.read(4)
```
se n. de caracteres não especificado, ler todo o arquivo

```
s = f.readline()
```
ler a próxima linha

```
f.close()
```
não esqueça fechar o arquivo no final

Fechado automático usando : `with open(…) as f:`

Bem comum: loop para ler as linhas de um arquivo de textos

```
for linha in f :
    # bloco que processa cada linha
```

Formatação de Cadeias

diretivas de formatação — valores a formatar

```
"model {} {} {}".format(x,y,r) → str
"{seleção:formatação!conversão}"
```
- **Seleção** :
  `2`
  `x`
  `0.nome`
  `4[chave]`
  `0[2]`
- **Formatação**:

enchimento *tabulação* *signo* *largura mín* . *precisão~largura máx* *tipo*

`<` `>` `^` `=` `+` `-` *espaço* `0` ao inicio para prencher com 0

inteiros: `b` binário, `c` caractere, `d` decimal (padrão), `o` octal, `x` or `X` hexa

reais: `e` o `E` exponencial, `f` ou `F` ponto fixo, `g` ou `G` geral (padrão), `%` percentagem

cadeia: `s` …

Exemplos:
```
"{:+2.3f}".format(45.7273)
→'+45.727'
"{1:>10s}".format(8,"toto")
→'      toto'
"{!r}".format("I'm")
→'"I\'m"'
```
- **Conversão** : `s` (texto legível) ou `r` (representação literal)