

# python™

**Beautiful** is better than ugly.  
**Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules.

Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it may be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

**Beautiful** is better than ugly.  
**Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules. Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it may be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

Sjsoft, <http://westmarch.sjsoft.com/2012/11/zen-of-python-poster/>

## PROGRAMAÇÃO E PYTHON

# Porquê Programar?

- Com ferramentas resolvem-se problemas
  - ▣ Aplicando soluções existentes
- Programando resolvem-se **novos** problemas
  - ▣ Ou velhos problemas de novas maneiras
- Tudo são bits e algoritmos
  - ▣ Som, Imagem, documentos, música, etc...

# Linguagens

- Linguagens são ferramentas
  - ▣ Um mecânico tem várias chaves
- Existem diferentes necessidades:
  - ▣ Aplicações
  - ▣ Páginas Web
  - ▣ Aplicações Móveis
  - ▣ Desenvolvimento rápido
  - ▣ Velocidade de execução
  - ▣ Compreensão
  - ▣ Etc...

# Porquê Python

- Java: aplicações, serviços, web, mobile
  - ▣ Desenvolvimento rápido
- Javascript: páginas e serviços web
- Linguagem interpretada
  - ▣ Não é necessário compilar código

# Python

---

- Python: aplicações, serviços, web, mobile
- Desenvolvimento muito rápido (prototipagem)
- Linguagem obriga a formatação rígida
  - ▣ “Hacks” são sempre formatados corretamente

# Python

- Nome: Monty **Python**'s Flying Circus
- Combina funcionalidades modernas
  - ▣ Encontradas no Java, C#, Ruby, C++, etc...
- Com um estilo conciso e simples

# Zen of Python

- Python possui um código de princípios
- Guiam a linguagem e os programas que a utilizam

```
$> python
```

```
>>> import this
```

# *Simple is better than complex*

- Só existem 31 palavras reservadas
  - ▣ Java: ~50
  - ▣ JavaScript: ~60 + ~111 (DOM)
  - ▣ C++: ~50
  - ▣ C#: ~80

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	



# *Beautiful is better than ugly*

- Indentação define um bloco
  - ▣ Sempre com espaço ou tabs (nunca ambos)
  - ▣ 4 espaços
- ENTER delimita fim de linha
- Nomes usam separador “\_”
  - ▣ Ex: processa\_ficheiro

# Python: Hello World! (mínimo)

Ficheiro hello.py

```
# File: hello.py  
  
print "hello world"
```

Consola

```
$> python hello.py  
  
hello world
```

# Variáveis

- Declaram-se sem tipo
  - Tipo dinâmico

```
# File: vars.py
```

```
a = 3
```

```
b = 5.2
```

```
print a * b
```

```
a = "var"
```

# Variáveis String

- Podem ser tratadas como os *arrays* em Java
- Não existe *char* (é uma *string* com 1 carácter)
- Tamanho dado por função *len*

```
a = "hello"  
b = "world"  
print a+" "+b  
print a[1]  
print a[1:4]  
print len(a)
```

```
hello world  
  
e  
  
ell  
  
5
```

# Variáveis String

- ❑ Concatenação com inteiros NÃO funciona
  - ❑ Necessário converter inteiros em String

```
r = 42
```

```
s = "A resposta para a vida, o Universo e \  
tudo mais é: "
```

```
print s + r
```

→ **TypeError: cannot concatenate 'str' and 'int' objects**

```
print s + str(r)
```

→ A resposta para a vida, o  
Universo e tudo mais é: 42

# Variáveis String

- Não existe *printf*
- Mas é possível formatar *strings*

```
r = 42
s = "A resposta para a vida, o Universo e \
    tudo mais é:"
print "%s %d" % (s, r)
```

A resposta para a vida, o Universo e tudo mais é: 42

# Condições

- Usam-se operadores “and”, “or”, “not” explícitos

```
ano = 2000
if (ano % 4==0 and ano % 100 != 0) or ano % 400== 0:
    bissexto = True
else:
    bissexto = False

if bissexto:
    ndias = 29
else:
    ndias = 28
```

# *Beautiful is better than ugly*

## **ERRADO**

```
if a == 3 and b == False: print "3"
```

## **CORRETO**

```
if a == 3 and not b:  
    print "3"
```



# Ciclos: For



```
for i in range(1,10):  
    print i
```

```
1  
2  
3  
...  
9
```

# Ciclos: Range

- Cria uma lista entre 2 valores

```
print range(1,10)
```

```
print range(10)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
[0,1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# Ciclos: While


```
a = 3
while a > 0:
    print a
    a = a - 1
```

```
3
2
1
```

# Funções

Declaração de função

Argumentos



```
def foo(name):  
    print "Olá: "+name
```

The diagram shows two orange arrows. One arrow points from the text 'Declaração de função' to the 'def foo(name):' line of the code. The other arrow points from the text 'Argumentos' to the '(name)' part of the same line.

```
foo("Pedro")
```

**Indentação define bloco**

# Funções

Declaração de função

Ciclo While

```
def factorial(x):  
    a = 1  
    while x > 0:  
        a = a * x  
        x = x - 1  
    return a
```

Declaração de variável  
e atribuição

**Indentação define bloco**

# Listas

- Python não possui *arrays* como o Java
- Lista é o mais semelhante

```
a = [1, 2, 3]
```

```
print a[1]
```

```
print len(a)
```

```
for v in a:
```

```
    print v
```

2

3

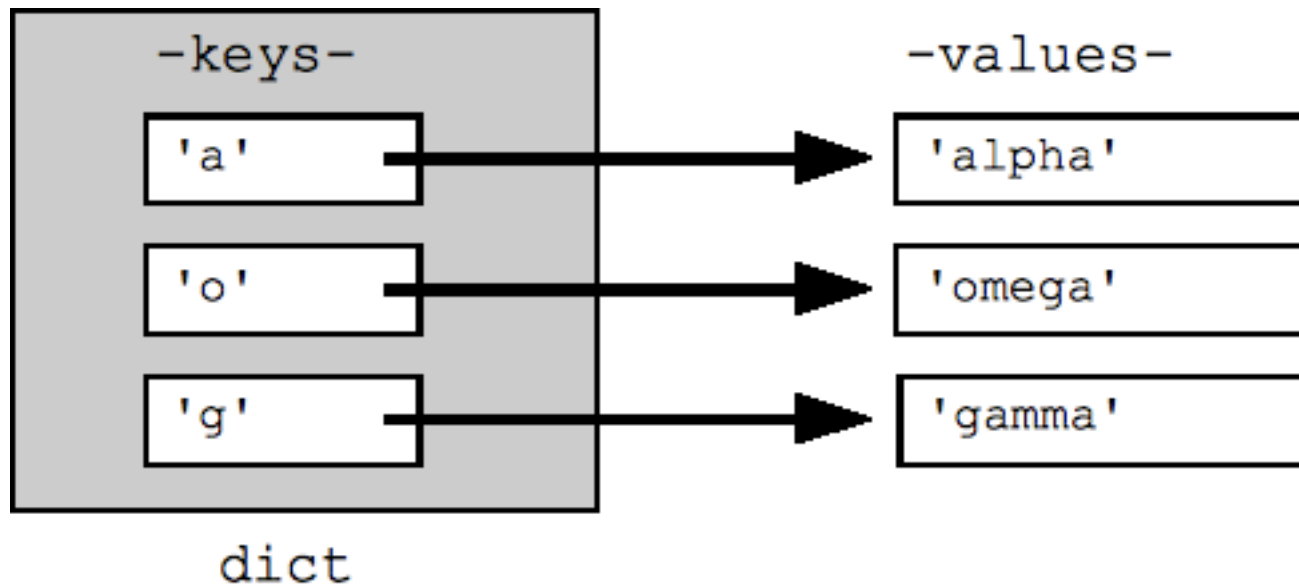
1

2

3

# Dicionários

- Estrutura que mapeia chave a valor
- Elementos não possuem ordem



# Dicionários

```
d = {"nome": "Pedro", "mec": 123, "turma": 0}  
d["turma"] = "TP5"  
print d["nome"]  
print d
```

```
Pedro  
{ 'mec': 123, 'nome': 'Pedro', 'turma': 'TP5' }
```



# Módulos

- Funcionalidades adicionais são fornecidas em módulos
- Adicionados ao programa com “*import*”
  - ▣ Semelhante ao Java
- Cada programa usa módulos conforme necessário

# Módulos

- Programa imprime o número e conteúdo dos argumentos passados
  - ▣ Argumentos presentes numa lista `sys.argv[]`
  - ▣ `sys.argv[0]` contém o nome do programa

```
import sys

print "Número: %d" % (len(sys.argv))
print "Valores: %s" % (str(sys.argv))
```

```
Número: 4
Valores: ['modules.py', 'a', 'b', 'c']
```

# Para Referência

- ❑ Python Docs: <http://docs.python.org/>
- ❑ Code Like a Pythonist:  
[http://python.net/~goodger/projects/pycon/2007/i  
diomatic/handout.html](http://python.net/~goodger/projects/pycon/2007/i<br/>diomatic/handout.html)
- ❑ Learn Python: <http://www.learnpython.org/>
- ❑ Think Python:  
<http://www.greenteapress.com/thinkpython/>