Rudimentos de Python

Para comezar

A linguaxe *Python* está deseñada para que os seus códigos de programa sexan fáciles de ler e entender. É unha grande linguaxe para iniciarse na programación con código, e a medida que se vai avanzando, afrontar proxectos cada vez máis complexos. Esta é a grande vantaxe de *Python*: a sintaxe é fácil e as amplas posibilidades que ofrece.

Nesta unidade abordaremos os rudimentos da linguaxe, vendo como se desenvolven en *Python* os elementos habituais, presentes noutros sistemas de programación.

Vinicius Depizzol en Wikimedia Commons.

Python icon done in the Tango! style

(GNU/GPL)

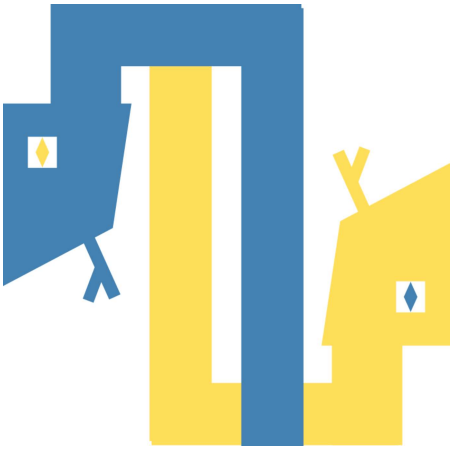
Obxectivos

- *Coñecer as características principais e as posibilidades da linguaxe de programación Python*
- *Interpretar programacións básicas en Python e ser quen de modificalas.*
- *Crear un programa básico con Python*

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Que é Python



notKlaatu en Openclipart. *Python Generic Application Logo* (Dominio público)

Python é unha linguaxe de programación por código de propósito xeral, cunha sintaxe moi sinxela e, inda así, con grandes posibilidades. É unha linguaxe moi interesante para aprender na medida en que podemos percibir a súa aplicación no mundo profesional.

Prográmase mediante o *IDE* de *Python* ou de calquera editor de texto como pode ser *Wordpad* en *Windows* ou *Gedit* en *Linux*. Tamén existen interfaces externas específicas que facilitan a creación de código.

Existen dúas versións, coas súas correspondentes actualizacións: *Python 2* ou *Python 3*. Ambas as versións de Python coexisten no tempo para garantir a maior compatibilidade posible con todas as ferramentas existentes na rede, sendo dúas versións vivas e actualizadas. De cara á aprendizaxe do usuario as diferenzas son mínimas.

Características fundamentais

\+ -

» **É unha linguaxe interpretada**

Execútase empregando un programa intermedio (intérprete), no canto de complilar directamente a unha linguaxe máquina coma nas linguaxes compiladas.

» **Orientada a obxectos**

É unha linguaxe de programación que utiliza compoñentes de software: obxectos. Os obxectos son compoñentes ou código de software que

conteñen en si mesmos tanto as súas características (campos) como os seus comportamentos (métodos).

» **É multiparadigma**

Soporta máis dun paradigma de programación. Permite programar empregando máis dun estilo de programación, coma por exemplo:

+ -

» » **Programación imperativa**

Paradigma de programación que describe a programación en termos de estado do programa e sentenzas que cambian o devandito estado. Os programas imperativos son un conxunto de instrucións que lle indican ao computador como realizar unha tarefa.

» » **Programación funcional**

Programación declarativa baseada no emprego de funcións matemáticas.

» **Emprega unha tipaxe dinámica**

Non é preciso declarar o tipo de dato que se vai conter unha determinada variable, senón que o seu tipo se declara no tempo de execución

» **Xestión automática de memoria**

Mediante a conta de referencias pódese establecer, cando non existe referencia algunha, bloques de memoria que se poden liberar.

» **Extensa biblioteca estándar**

A súa biblioteca (*STL*), así coma os módulos aportados por terceiros cobren practicamente todas as tarefas.

» **Integrable dentro das aplicacións**

Pódese integrar noutras aplicacións coma unha interface de *scripting*.

Aplicacións

Ao ser unha linguaxe multipropósito e altamente portable, *Python* utilizouse para desenvolver:

- Aplicacións de escritorio.
- Aplicacións web.
- Análise de datos.
- Administración de servidores.
- Seguridade e análise de penetración.
- Cómputo na nube.
- Cómputo científico.
- Análise de linguaxe natural.
- Visión artificial.
- Animación, videoxogos e imaxes xeradas por computadora.
- Aplicacións móbiles.

Mantemento, desenvolvemento e licenza

Python é administrado pola *Python Software Foundation*. Posúe unha licenza de código aberto, denominada *Python Software Foundation License*, que é compatible coa Licenza pública xeral de *GNU* a partir da versión **2.1.1**, e incompatible en certas versións anteriores.

Python Software Foundation. *Python Software Foundation* (PSF LICENSE)

A ter en conta: Por que Python?

A seguinte cita pode axudarnos a compender a razón de que en moitas ocasións se escolla a linguaxe *Python.* A continuación temos o vídeo *Por que aprender a programar con Python?* que desenvolve a mesma idea.

Por que aprender a programar con Python?

Por que Python?

> *Python é unha linguaxe que todo o mundo debería coñecer. A súa sintaxe simple, clara e sinxela; o tipado dinámico, o xestor de memoria, a gran cantidade de librerías dispoñibles e a potencia da linguaxe, entre outros, fan que desenvolver unha aplicación en Python sexa sinxelo, moi rápido e, o que é máis importante, divertido. A sintaxe de Python é tan sinxela e próxima á linguaxe natural que os programas elaborados en Python parecen "pseudocódigo". Por este motivo trátase ademais dunha das mellores linguaxes para comezar a programar.*
>
> *Raúl González Duque en "Python para todos". [Documento pdf en liña] dispoñible en: https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf. (Tradución libre)*

https://www.youtube.com/embed/9r2wF93vOkM

José Domingo Muñoz en OpenWebinars. *Por qué aprender a programar con python?* (CC BY-SA)

Orixe de Python

Python foi creado a finais dos anos oitenta por Guido van Rossum no *Centro de Matemáticas e Tecnoloxías da Información* (CWI, Centrum Wiskunde & Informática), nos Países Baixos, como sucesor do linguaxe de programación *ABC,* capaz de manexar excepcións e interactúan co sistema operativo *Amoeba.3*.

Wikimedia Commons. *Monty Python settanta* (Dominio público)

O nome da linguaxe provén do cariño do seu creador polos humoristas británicos *Monty Python*. Van Rossum é o autor principal de *Python,* cun fundamental papel na dirección que tomou a linguaxe

Fonte: Colaboradores de Wikipedia. *Python* [en liña]. Wikipedia, La enciclopedia libre, 2018. Dispoñible en <https://es.wikipedia.org/w/index.php?title=Python&oldid=107334378>.

Rudimentos de Python

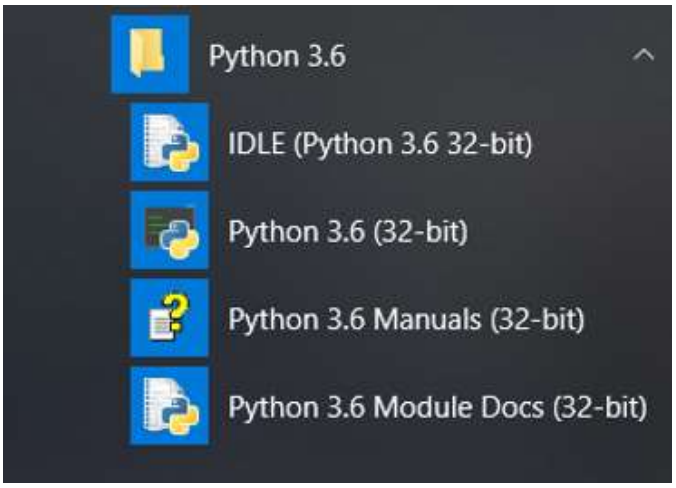
Primeiros pasos

Manuel Torres Búa. Primeiros pasos. (CC BY-SA). A partir de imaxe: Clker-Free-Vector-Images en Pixabay. *footprints*. (CC0)

Descarga e instalación do programa

Compoñentes de *Python*

O programa preséntase con catro compoñentes



Manuel Torres Búa. *Compoñentes de Python* (Dominio público)

1. ***IDLE*** (*integrated development and learning environment)*, é o editor de *Python* por defecto. Poderiamos utilizar moitos outros pero é ideal para comezar.
2. **Python *3.6*** é o intérprete… É o programa que *executará* as ordes que lle escribamos.
3. **Manuais.** É importante consultalos se queremos afondar máis.
4. **Documentos de Módulos.** Permiten ampliar as posibilidades da linguaxe.

Rudimentos de Python

Primeiros pasos

Manuel Torres Búa. Primeiros pasos. (CC BY-SA). A partir de imaxe: Clker-Free-Vector-Images en Pixabay. *footprints*. (CC0)

Descarga e instalación do programa

Compoñentes de *Python*

Desenvolvéndose na contorna de *Python*

Comezando a escribir código

Ao comezar a programar, debemos ter presentes algunhas características básicas da linguaxe *Python* que a diferencian doutras linguaxes de programación.

\+ -

» `(>>>)` ***Prompt***

É o punto que nos sinala o *IDLE* para introducir instrucións.

Sen punto e coma

A instrución se pecha sen poñer nada, nin o punto e coma que se pon noutras linguaxes. No *IDLE* para executar a instrución o único que facemos e pulsar *enter* ↵

» `(___)` **Cada instrución nunha liña**

Un programa de *Python* está formado por liñas de texto. Recoméndase que cada liña conteña unha única instrución.

```
>>> print ("Ola")
Ola
>>> print ("Que tal estás?")
Que tal estás?
>>>
```

» (;) **Punto e coma**

É posible incluír máis dunha instrución nunha liña, pero non é recomendable. Só se utiliza esta características en programas moi complexos. Para facelo só hai que separar as instrucións con punto e coma(;)

```
>>> print ("Ola"); print ("Que tal estás?")
Ola
Que tal estás?
>>>
```

» (\) **Separación de liñas**

Por motivos de lexibilidade recoméndase que as liñas non superen os 79 caracteres. Se unha instrución supera esa lonxitude, pódese dividir en varias liñas usando o carácter contrabarra (\):.

```
>>> #Declaramos avariable nome
>>> nome="O meu nome é Pepe"
>>> #chamamos á variable
>>> nome
'O meu nome é Pepe'
>>> #facemos o mesmo con separación de liñas
>>> nome="o meu nome é\Pepe"
>>> #chamamos de novo á variable
>>> nome
'o meu nome é Pepe'
>>>
```

Espazos

Os elementos da linguaxe sepáranse por espazos en branco (normalmente, un), aínda que nalgúns casos non se escriben espazos:

- Entre os nomes das funcións e a paréntese
- Antes dunha coma (,)
- Entre os delimitadores e o seu contido (paréntese, chaves, corchetes ou comiñas)

Os espazos non son significativos, é dicir, dá o mesmo un espazo que varios, excepto ao principio dunha liña. Os espazos ao principio dunha liña (o sangrado) indican un nivel de agrupamento. Unha liña non pode conter espazos iniciais, a menos que forme parte dun bloque de instrucións ou dunha instrución dividida en varias liñas.

Comentarios

Para agregar comentarios na programación basta con poñer o cancelo (#) despois do prompt. Todo o que escribamos a continuación serán comentarios que non interfiren na programación. Se o que desexamos é agregar textos de comentarios en varias liñas, debemos pechar o texto entre comiñas ou parágrafos triplos.

Os textos de comentario en varias liñas pódense empregar para xerar documentación de axuda: son sos denominados *Docstrings*.

Identation

En moitas ocasións é necesario delimitar bloques de código para interpretar ben os programas. O que facemos para identificar un bloque é poñer a primeira instrución pegada á marxe esquerda, e as seguintes liñas que dependen da primeira sangradas á dereita. (Vaino facer a consola directamente)

```
def fromlist(cls, v):
        if not isinstance(v, list):
            raise TypeError
        inst = cls()
        inst.v = v
        return inst
```

Rudimentos de Python

Primeiros pasos

Manuel Torres Búa. Primeiros pasos. (CC BY-SA). A partir de imaxe: Clker-Free-Vector-Images en Pixabay, *footprints*. (CC0)

Descarga e instalación do programa

Compoñentes de *Python*

Desenvolvéndose na contorna de *Python*

Comezando a escribir código

As cores do código *Python*

- **As palabras reservadas de *Python* (as que forman parte da linguaxe) móstranse en cor laranxa.**
- **As cadeas de texto móstranse en verde.**
- **Os resultados das ordes escríbense en azul.**
- **As mensaxes de erro móstranse en vermello.**
- **As funcións móstranse en púrpura.**

(Os comentarios tamén aparecerán en vermello)

A ter en conta: Tipos de arquivo en Python

Rudimentos de Python

Palabras reservadas

As palabras reservadas (*keywords*) son aquelas que non se poden empregar para nomear variables, xa que teñen unha función específica en *Python*. Son o núcleo da linguaxe *Python*.

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Podemos ver a función de cada palabra no seguinte documento:

- *Palabras reservadas del lenguaje* en "Recursos Python". Dispoñible en: https://recursospython.com/guias-y-manuales/palabras-reservadas-del-lenguaje/

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Datos

En *Python* todos os seus elementos son obxectos. Tamén, unha vez que se identifican, os datos convertense obxectos relacionados co tipo ao que pertencen.

Manuel Torres Búa. *Datos* (CC BY-SA) a partir de imaxes de IO-Images en Pixabay.(CC0)

Numérico

\+ -

» **Enteiros (*int long*)**

Os números enteiros son aqueles que non teñen decimais, tanto positivos coma negativos (ademais do cero). En *Python* pódense representar mediante o tipo ***int*** (de *integer*, enteiro) ou o tipo ***long*** (longo). A única diferenza é que o tipo *long* permite almacenar números máis grandes. É aconsellable non utilizar o tipo long a menos que sexa necesario, para non malgastar memoria.

```
Decimal: 65, 14
Binario: 0b010011 , 0b1101
Hexadecimal: 0x18 , 0x3cf4
Octal: 030 , 074
```

» **Reais (*float*)**

Os números reais son o conxunto de obxectos *flotantes (**float**)*. Para representar un número real en *Python* escríbese primeiro a parte enteira,

seguido dun punto e por último a parte decimal.(real = 0.2703). Tamén se pode utilizar notación científica, e engadir unha *e* (de expoñente) para indicar un expoñente en base 10. Por exemplo: real = 0.1e-3. Hai que ter coidado co uso da coma: un número decimal con coma, *Python* o interpreta coma unha parella de números.

```
3.141595
28
-62.5974
```

» **Números complexos (*complex*)**

Python pode facer cálculos con número complexos. Aparece a parte real, o operador "+" e a parte imaxinaria acompañada da letra "j" ao final.

```
6.32 + 45j
0.117j
(2 + 0j)
1j
```

Lóxico

O tipo de datos lóxico, tamén chamado *booleano* é un tipo que só ten dous valores posibles: **verdadeiro** e **falso**. Este tipo de dato úsase para realizar operacións lóxicas sobre os datos, tomar decisións e controlar a execución dos programas.

- Se a expresión lóxica é verdadeira, o resultado é ***True*** (con maiúsculas ao comezo).
- Se a expresión lóxica non é verdadeira, o resultado é ***False*** (en maiúsculas ao comezo).

False é numericamente igual a **0**. Calquera outro número é igual a Verdadeiro eo seu valor predeterminado é **1**.

Manuel Torres Búa. Verdadeiro e falso. (CC BY-SA). A partir de imaxe: Geralt en Pixabay. *Verdadeiro e falso.* (CC BY-SA.)

Carácter

O tipo de datos de carácter permite procesar letras, números, símbolos e ideogramas ao asociar un código numérico a cada carácter que se desexa representar. *Python 3* utiliza a codificación ***UTF-8*** por defecto.

Os caracteres agrúpanse en cadeas de caracteres. As cadeas son secuencias de caracteres entre comiñas (" ") ou apóstrofos (' ')

A ter en conta

Hai unha serie de funcións que especialmente serven para tratar os diferentes tipos de datos

Funcións relativas aos tipos de datos

+ -

» ***type()***

Identifica o tipo de dato dunha variable.

```
1 type("Ola")
2 str
3 type(12)
```

```
int
type(23j)
complex
```

» ***str()***

Transforma un obxecto (compatible) nunha cadea de caracteres.

» ***int()***

Transforma un obxecto (compatible) nun obxecto tipo *int* (enteiro).

Pode converter obxectos de tipo ***str*** a un número enteiro.

Trunca os obxectos de tipo ***float*** á parte enteira.

True é convertido en 1 e False en 0.

(Non é compatible con obxectos tipo complex)

» ***float()***

Transforma a un obxecto compatible a un de tipo ***float***.

Pode converter obxectos de tipo ***str*** que conteñan representen correctamente a un número real.

É compatible cos obxectos tipo ***int***.

True é convertido en 1.0 e ***False*** en 0.0.

(Non é compatible con obxectos tipo ***complex)***

» ***complex()***

Transforma a un obxecto compatible a un de tipo ***complex***.

Converte obxectos de tipo ***str*** a un número real.

Transforma a un obxecto de tipo ***complex*** a un par de números xa sexan ***int*** ou ***float***. Se só se dá un número ***int*** ou ***float***, este será identificado como o compoñente real e o compoñente complexo será 0 j.

» ***bool()***

Transforma en boloneado a un obxecto

O 0 é igual a ***False***.

Calquera outra cousa diferente de 0 é ***True***.

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Expresións e Declaracións

Unha **expresión** é unha combinación de valores, operadores, funcións e métodos que dan como resultado un valor nunha única liña.

```
>>> 1+1
2
>>>
>>> 45>=11
True
>>>
>>> "carro".upper()
'CARRO'
```

As **declaracións** son unidades de código que o interprete de *Python* pode executar. En realidade unha declaración é un tipo de expresión. O intérprete de *Python* permite executar múltiples expresións nunha soa, separándoas por punto e comas ";". Neste caso, só se despregará o resultado da última expresión executada.

```
>>> a = 3; b = a * 4.5; b; a + 5
13.5
8
>>>
```

Manuel Torres Búa. *Expresións* (CC BY-SA). A partir da imaxe: Pexels en Pixabay. *coding.*. *(CC0)*

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operador	Descrición
+	Suma
-	Resta
-	Negativo
***	Multiplicación
**	Expoñente
/	División
//	División enteira
%	Residuo

Regras de precedencia en operacións aritméticas.

Os operadores gardan as seguintes regras de precedencia seguindo unha secuencia de esquerda a dereita:

1. Parénteses.
2. Expoñente.
3. Multiplicación.
4. División.
5. Suma.
6. Substración.

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operador	Descrición
+	Concatenación
\	Repetición

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operador	Avalía
==	*a* == *b* ¿a igual a b?
!=	*a != b* ¿a distinta de b?
>	*a* > *b* ¿a maior que b?
<	*a* < *b* ¿a menor que b?
>=	*a* >= *b* ¿a maior ou igual que b?
<=	*a* <= *b* ¿a menor ou igual que b?

Operadores lóxicos

Operadores de pertenza

Operadores de asignación

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operadores lóxicos

Operador	Avalía...
or	*a or b* Cúmprese a ou b?
and	*a and b* Cúmprese a e b?
not	*not x* Contrario a x

Operadores de pertenza

Operadores de asignación

Operadores de bits

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operadores lóxicos

Operadores de pertenza

Operador	Avalía...
in	*Un obxecto dentro de outro?*
not in	*Un obxecto non está dentro de outro*

Operadores de asignación

Operadores de bits

Operadores de identidade

Función eval()

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operadores lóxicos

Operadores de pertenza

Operadores de asignación

Empréganse para ligar un obxecto ou valor cun nome

Operador	Descrición	Exemplo
=	Asignación simple	*x = y*
+=	Suma	*x += y* equivale a *x = x + y*
-=	Resta	*x -= y* equivale a *x = x - y*
*=	Multiplicación	*x *= y* equivale a *x = x * y*
**=	Expoñente	*x ** = y* equivale a *x = x ** y*
/=	División	*x /= y* equivale a *x = x / y*

Operador	Descrición	Exemplo
//=	División enteira	*x* //= *y* equivale a *x* = *x* // *y*
%=	Residuo de división	*x* %= *y* equivale a *x* = *x* % *y*

Operadores de bits

Operadores de identidade

Función eval()

Fonte: "*Tipos de datos básicos y operadores*", *pythonista.mx* [En liña]. Dispoñible: https://pythonista.io/cursos/py101/tipos-de-datos-basicos-y-operadores.. (CC BY SA)

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operadores lóxicos

Operadores de pertenza

Operadores de asignación

Operadores de bits

Cálculos que implican a cada *bit* que conforma un número binario.

Operador	Descrición
&	AND
|	OR
^	XOR
<<	Mover x bits á esquerda
>>	Mover x bits á dereita

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Operadores aritméticos

Operadores para *str* (cadeas)

Operadores de relación

Operadores lóxicos

Operadores de pertenza

Operadores de asignación

Operadores de bits

Operadores de identidade

Permiten avaliar se un identificador refírese exactamente ao mesmo obxecto ou pertence a un tipo.

Operador	Avaliación
is	*a is b* Equivale a *id(a) == id(b)*
is not	*a is not b* Equivale a *id(a) != id(b)*

Por exemplo:

```
a = 45
b = 45
a is b
True
type("Ola") is str
True
True is 1
False
```

Función eval()

Fonte: “*Tipos de datos básicos y operadores*”, *pythonista.mx* [En liña]. Dispoñible: https://pythonista.io/cursos/py101/tipos-de-datos-basicos-y-operadores.. (CC BY SA)

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Operadores

Un operador é un símbolo, un carácter ou unha combinacion de varios caracteres que forman parte das regras sintácticas de *Python* e permiten realizar unha determinada operación entre un ou máis datos (operandos) que produce un resultado. Temos varios tipos.

Mediante eval podemos avaliar se un obxecto tipo *str* (cadea de caracteres) coma se fose unha expresión. Se o texto a avaliar non é válido daranos unha mensaxe de erro. Por exemplo:

```
1 eval("12 * 300")
2 3600
3
```

```
4 eval("12 > 5")
5 True
```

Fonte: “*Tipos de datos básicos y operadores*”, *pythonista.mx* [En liña]. Dispoñible: https://pythonista.io/cursos/py101/tipos-de-datos-basicos-y-operadores.. (CC BY SA)

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Funcións

Geralt en Pixabay. *learn* (CC0)

Unha función é un subproceso ou subalgoritmo formado por unha secuencia de instrucións cunha tarefa específica. Ás veces, nun determinado programa é necesario efectuar unha mesma tarefa en distintos puntos do programa. A miúdo, en diferentes programas tamén se repiten moitas veces. Para evitar ter que volver escribir unha e outra vez as mesmas instrucións, case todas as linguaxes de programación permiten agrupar porcións de programa e reutilizalos nun mesmo programa ou en diferentes. Son as **subrutinas**. Se unha subrutina se vai utilizar nun único programa, adóitase definir no propio programa. Pero cando unha subrutina emprégase en varios programas, non é necesario repetila en cada programa (perderíanse algunhas das vantaxes comentadas anteriormente), senón que se pode incluír nun ficheiro aparte ao que os programas poden acceder para recuperar as subrutinas. Estes ficheiros reciben o nome de **bibliotecas** (en español adóitase dicir moito librería, traducindo incorrectamente o termo inglés library). En *Python* utilízase o termo **función** para referirse ás **subrutinas** e o termo **módulo** para referirse ás **bibliotecas.**

A ter en conta: definindo funcións

Aspectos relacionados coa sintaxe nas funcións

Funcións en Python

\+ -

» **Definindo funcións**

A sintaxe máis simple para un **bloque de código dunha función** é a seguinte:

```
def functionName ( ) :
    ......
    instrucións de funcións
    ......
```

Exemplo:

```
def sauda():
 print ("Ola")

print ("Vou saudar")
sauda()
print ("Saudo de novo")
sauda()
```

Obteno como resultado:

```
Vou saudar
Ola
Saudo de novo
Ola
```

» Parámetros dunha función

Cando defino un parámetro nunha función (cun nome entre parénteses), podo usalo nela coma unha variable. O nome que eliximos realmente non importa, así que mellor empregar nomes significativos. Cando invocamos a función, debemos darlle un valor entre parénteses, que será o valor que usa *Python* para executar a función. Por exemplo:

```
def suma(a,b):
    return a + b
print ("A suma vale...")
print (suma(15, 5))
```

» Argumentos

Os argumentos son os valores que empregan as funcións. Podemos poñer varios valores se os separamos mediante comas. Exemplo:

```
def escribe_media():
    media = (a + b) / 2
    print(f"Aa media de {a} e {b} é: {media}")
    return

a = 3
b = 5
escribe_media()
print("Fin do programa")
```

```
Aa media de 3 e 5 é: 4.0
Fin do programa
```

Referencia: Funcións integradas

Python dispón dunha serie de funcións sempre dispoñibles. Son as *built-in functions.* Inserimos o documento onde se explican as funcións integradas que veñen na documentación oficial de *Python.* (En inglés)

Ocultar a retroacción

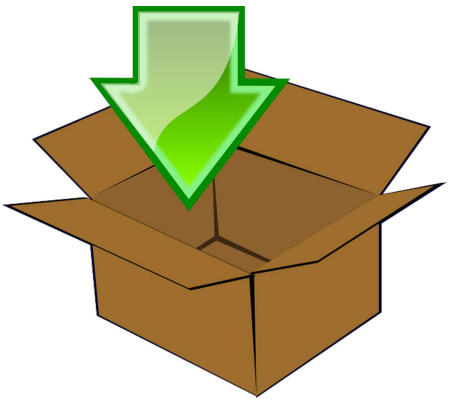
https://docs.python.org/3/library/functions.html

The Python Standard Library. *Built-in Functions* (Tódolos dereitos reservados)

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Variables



Clker-Free-Vector-Images en Pixabay. *Storage* (CC0)

Diciamos noutras unidades que para moitas linguaxes de programación as variables poden entenderse como *caixas* nas que se almacenan os datos, pero en *Python* as variables son ***etiquetas*** que permiten a referencia aos datos (que se almacenan en *caixas* chamadas **obxectos**).

Python é unha **linguaxe de programación orientada a obxectos** e o seu modelo de datos tamén está baseado en obxectos.

Para cada dato que aparece nun programa, *Python* crea un obxecto que o contén. Cada obxecto ten:

- Un identificador único (un número enteiro, diferente para cada obxecto). O identificador permite a *Python* facer referencia ao obxecto inequivocamente.
- Un tipo de datos (enteiro, decimal, cadea de caracteres, etc.). O tipo de datos permite a *Python* saber que operacións se poden facer cos datos.
- Un valor (os datos en si).

Deste xeito, as variables en *Python* non gardan os datos, senón que son nomes simples para poder referirse a eses obxectos. Se escribimos a instrución ...

```
a = 2
```

Noutras linguaxes estamos indicando que creamos a variable a e introducimos o valor 2 nesa caixa a. En *Python* o significado é diferente:

- Crear o obxecto "2". Ese obxecto terá un identificador único asignado no momento da creación e mantido durante todo o programa. Neste caso, o obxecto creado será de tipo enteiro e manterá o valor 2.
- Asocia o nome *a co* número enteiro obxecto 2 creado.

\+ -

» **Definir variables**

As variables de *Python* créanse cando están definidas por primeira vez, é dicir, cando se lle atribúe un valor por primeira vez. Para asignar un valor a unha variable, úsase o

operador de igualdade (=). Á esquerda da igualdade escríbese o nome da variable e á dereita o valor que desexa dar á variable.

» Nomear variables

Aínda que non sexa obrigatorio, recoméndase que o nome da variable estea relacionado coa información almacenada nel, de forma que sexa máis doado comprender o programa. Se o programa é trivial ou mentres se está a escribir un programa, isto non parece ser moi importante, pero se consultamos un programa escrito por outra persoa ou escrito por un mesmo pero hai algún tempo, será moito máis doado comprendelo se os nomes son correctos.

No *IDLE* invocamos a variable simplemente escribindo o seu nome

```
>>> a = 2
>>> a
2
```

Tamén se adoita empregar certos nomes de variables nalgúns casos, como veremos máis tarde, pero tampouco é obrigatorio. Como normas xerais debemos respectar o seguinte:

- O nome dunha variable debe comezar cunha letra ou un guión baixo (_) e pode continuar con máis letras, números ou guións baixos.
- Os nomes de variables non poden incluír espazos en branco.
- Os nomes das variables poden conter calquera carácter alfabético (aqueles do alfabeto inglés, pero tamén ñ, ç ou vogais acentuadas), aínda que se recomenda usar só os caracteres do alfabeto inglés.
- Os nomes das variables poden ser maiúsculas, pero hai que ter en conta que *Python* distingue entre maiúsculas e minúsculas (en inglés, *Python* é "sensíbel a maiúsculas e minúsculas").
- Cando o nome dunha variable contén varias palabras, é recomendable separalos con guións baixos para facilitar a lectura, aínda que tamén se usa a notación *CamelCase* , onde as palabras non están separadas pero comezan con maiúsculas (excepto a primeira palabra).
- As palabras reservadas (*keywords*) da linguaxe (aquelas que *IDLE* escribe en laranxa) están prohibidas como nomes de variables.
- Os nomes das funcións incorporadas poden usarse como nomes de variables, pero é mellor non facelo porque entón a función xa non pode utilizarse como tal

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Secuencias: Tuplas, Listas e Rangos

+ -

» **Tuplas**

En *Python,* unha **tupla** é un conxunto ordenado e inmutable de elementos do mesmo tipo ou diferente. As *tuplas* están representadas escribindo os elementos entre parénteses e separados por comas.

```
>>> (1, "a" , 3.14)
 (1, 'a', 3.14)
```

» **Listas**

As *listas* son conxuntos de elementos ordenados (números, cadeas, listas, etc.). As *listas* están delimitadas por corchetes ([]) e os elementos están separados por comas.

As listas poden conter elementos do mesmo tipo:

```
>>> primos = [2, 3, 5, 7, 11, 13]
>>> díasLaborables = [ "Luns" , "Martes" , "Mércores" , "Xoves"
```

Ou poden conter elementos de diferentes tipos:

```
>>> data = [ "Luns" , 27, "outubro" , 1997]
```

Ou poden conter listas:

```
>>> películas = [[ "Paths of Glory" , 1957], [ "Hannah e as súas
```

» Rangos

En *Python* un rango (***range***) es un tipo de datos. O tipo ***range*** é una **lista inmutable de números enteiros en sucesión aritmética.**

O tipo de intervalo é unha **lista sen cambios de enteiros en secuencia aritmética .**

- Inmutable significa que, a diferenza das listas, os intervalos non se poden modificar.
- Unha sucesión aritimética é unha sucesión na que a diferenza entre dous termos consecutivos é sempre a mesma.

Un intervalo é creado chamando ao tipo de datos cun, dous ou tres argumentos numéricos, coma se fose unha función.

En *Python 2*, o intervalo () foi considerado unha función, pero en *Python 3* non se considera unha función, senón un tipo de datos, aínda que se usa como se fose unha función.

O tipo de rango () cun único argumento escríbese intervalo (n) e crea unha lista inmutable de *n* enteiros consecutivos que comeza en 0 e remata en *n - 1* .

Para ver os valores do intervalo (), é necesario convertelo en lista mediante a función list ().

```
>>> x = range(10)
>>> x
intervalo (0, 10)
>>> lista (x)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> intervalo (7)
intervalo (0, 7)
Lista >>> ( intervalo (7))
[0, 1, 2, 3, 4, 5, 6]
```

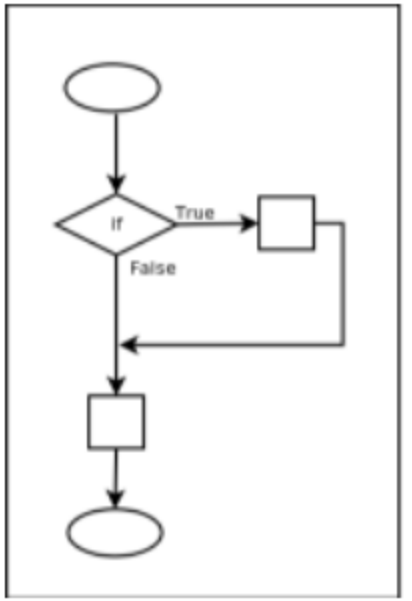
En resumo, os tres argumentos del tipo *range* (*m, n, p*) son:

- *m*: o valor inicial
- *n*: o valor final (que non se acada nunca)
- *p*: o paso (a cantidade que se avanza cada vez)

if ... (Se...)

If sempre avalía unha expresión lóxica e no caso de que a expresión teña resultado ***True*** , o código que está indentado (sangrado) a continuación executarase . No caso de que a declaración resulte ***False*** , o intérprete ignorará o bloque de código sangrado e continuarase coa seguinte instrución que exista despois dese bloque de código. O código sería da seguinte forma:

```
<fluxo principal>
...
...
if <expresión lóxica>:
     <bloque inscrito a if>
<fluxo principal>
```



Fundación Plone y amigos. *If Simple* (GNU/GPL)

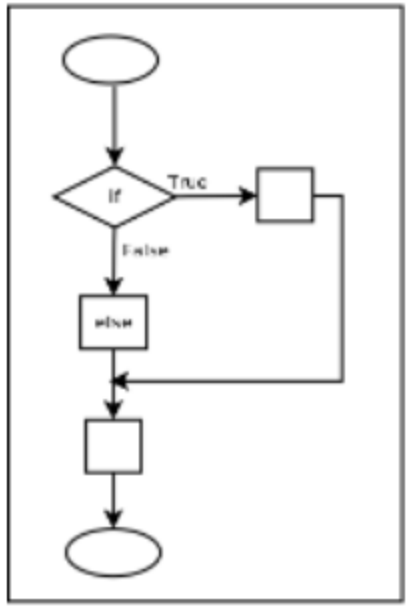
Vémolo no seguinte exemplo:

```
numero = int(input("Escribe un número maior que 10: "))
if numero <= 10:
    print("¡Díxenlle que escriba un número maior que 10!")
print(f"Escribiu o número {numero}")

========== RESTART: C:/Users/proba/proba.py ==========

Escribe un número maior que 10: 9
¡Díxenlle que escriba un número maior que 10!
Escribiu o número 9
```

if...else (Se...senón....)

A estrutura de control *if ... else* permite que un programa execute un ou varios bloques de código dependentes cando se cumpre unha determinada condición (*true*). Se non se cumpre o programa continúa

```
<fluxo principal>
...
...
if <expresión lóxica>:
     <bloque inscrito a if>
else:
    <bloque inscrito a else>
<fluxo principal>
```



Fundación Plone y amigos. *If...else* (GNU/GPL)

A primeira liña contén a condición para avaliar. Esta liña debe rematar sempre por dous puntos (:). A continuación ven o bloque de ordes que se executan cando a condición se cumpre (é dicir, cando a condición é verdadeira). Este bloque debe ir sangrado. Ao escribir dous puntos (:) ao final dunha liña, *IDLE* sangrará automaticamente as liñas seguintes.

Despois aparece a liña coa orde else, que indica a *Python* que o bloque que ven a continuación tense que executar cando a condición non se cumpra (é dicir, cando sexa falsa). Esta liña tamén debe rematar sempre por dous puntos (:). A liña coa orde else non debe incluír máis nada que o else e os dous puntos. En último lugar está o bloque de instrucións sangrado que corresponde ao else.

Vemos un exemplo:

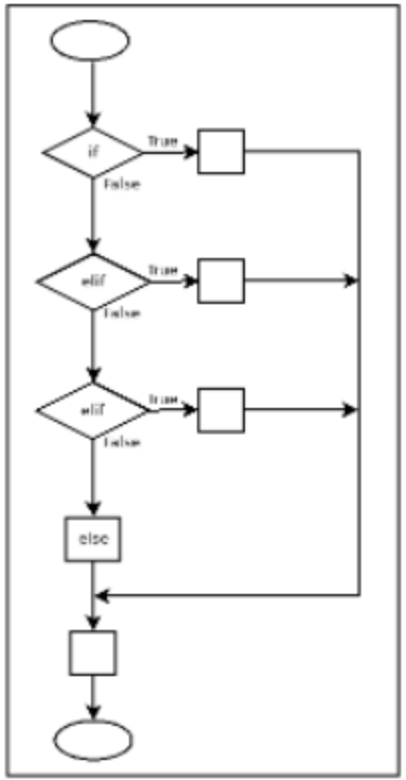
```
contrasinal = input ("introduce o contrasinal: ")
if contrasinal == "swordfish":
    print ("contrasinal aceptada")
else:
    print("Alerta intrusos")
print("Que teña un bo día")

========== RESTART: C:/Users//proba.py ==========
introduce o contrasinal: swordfish
contrasinal aceptada
Que teña un bo día
>>>
```

Introdúcese o valor da variable *contrasinal* por teclado mediante unha función *input*. Se o contrasinal é a palabra *swordfish* imprímese en pantalla a mensaxe "Contrasinal aceptada". Se non o é, aparece en pantalla a mensaxe "Alerta intrusos". Unha vez aceptada ou rexeitada o contrasinal, chégase á instrución que presenta en pantalla a mensaxe "Teña un bo día".

Unha sentencia condicional pode contar á súa vez outra ou varias sentencias condicionais dependentes nunha estrutura que, coma noutras linguaxes, denomínase ***Estrutura aniñada***.

if...elif...else

É posible avaliar máis dunha expresión lóxica mediante o uso de *elif*. A estrutura de control *if ... elif ... else ...* permite encadear varias condiciones. *elif* é una contracción de *else if*. No caso de que exista máis dunha expresión lóxica que dea como resultado *True*, *Python* executará soamente o código delimitado pola primeira na que ocorra. No caso de que ningunha das condicións dea resultado *True* pódese utilizar else ao final da estrutura.



Fundación Plone y amigos. *If...elif...else*

(GNU/GPL)

A orde en *Pyhton* escríbese así:

```
<fluxo principal>
...
...
if <expresión lóxica>:
     <bloque inscrito a if>
elif <expresión lóxica 1>:
     <bloque inscrito a elif>
elif <expresión lóxica 2>:
     <bloque inscrito a elif>
...
...
elif <expresión lóxica n>:
     <bloque inscrito a elif>
else:
    <bloque inscrito a else>
<fluxo principal>
```

Vemos un exemplo:

```
idade= int(input("Cal é a súa idade? "))
if idade < 10:
    print("Ten un desconto do 50%")
elif idade < 17:
    print("Ten un desconto do 25%")
else:
    print("Non ten vostede desconto algún")

========== RESTART: C:/Users/proba.py ==========
Cal é a súa idade? 8
Ten un desconto do 50%
>>>
========== RESTART: C:/Users/proba.py ==========
Cal é a súa idade? 15
Ten un desconto do 25%
>>>
========== RESTART: C:/Users/proba.py ==========
Cal é a súa idade? 33
Non ten vostede desconto algún
```

Este programa indica segundo a idade a tarifa que se lle aplica. Así para menores de 10 anos existe un desconto do 50%, de 10 a 16 anos do 25% e para o resto das idades non existe desconto.

Rudimentos de Python

Estruturas de control

Coma noutras linguaxes, é fundamental coñecer as estruturas de control de fluxo de datos de *Python* para case calquera programa que desexemos crear. Recordamos que en estruturas de control incluíamos as estruturas **secuenciais**, as **condicionais** e as **repetitivas.**

\+ -

» **Estruturas Secuenciais**

Manuel Torres Búa. *Secuencia* (CC BY-SA)

Unha estrutura secuencial, como sabemos, é aquela na que o fluxo de información do programa realízase en etapas consecutivas. Non nos pararemos neste tipo xa que podemos afirmar que inclúe calquera posibilidade de programación con *Python* sempre que non se altere sucesión ordenada do fluxo de información

» **Estruturas condicionais**

» **Estruturas repetitivas**

As estruturas repetitivas son aquelas nas que hai presenza de *bucles* ou *iteracións*. O programa vai repetir un ou varios bloques de código, ben un número determinado de veces, ben en función do cumprimento ou non dunha condición.

for...in

Con *for* pódese crear un ciclo que repite un ou varios bloques de instrucións unha cantidade preestablecida de veces. O bloque de instrucións que se repite adóitase denominar o **corpo** do ciclo e cada repetición adoita denominarse ***iteración***. A construción usa unha variable que en cada ciclo toma o valor dun novo elemento da lista da cláusula ***in***. O ciclo finalizará cando se acade o último elemento da lista ou cando o seu final estea forzado pola palabra ***break***

A sintaxe dun *for...in* é o seguinte:

```
for variable in elemento iterable (l
    corpo do bucle
```

Vemos un sinxelo exemplo:

```
print("Inicio")
for i in [0, 1, 2]:
    print("Chao ", end="")
print()
print("Fin")<br>

========== RESTART: C:/Users/proba.py ==========
Inicio
Chao Chao Chao
Fin
>>>
```

Podemos definir a variable ou empregar outra xa definida no programa. O número de repeticións o delimita *in,* e o corpo do bucle poden ser elementos illados, listas, cadeas, rangos,...

Ás veces podemos empregar variables que conteñen o número de veces que sucede algo (contadores) ou os valores que se acumulan (acumuladores).

Exemplo de contador:

```
print("Inicio")
conta = 0
for i in range(1, 6):
    if i % 2 == 0:
        conta = conta + 1
print(f"Dende 1 ata 5 hai {conta} múltiplos de 2")

====================================
Inicio
Dende 1 ata 5 hai 2 múltiplos de 2
```

Exemplo de acumulador:

```
print("Inicio")
suma = 0
for i in [1, 2, 3, 4]:
    suma = suma + i
print(f"A suma dos números de 1 a 4 é {suma}")

=========================
inicio
A suma dos números de 1 a 4 é 10
```

while

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0

Rudimentos de Python

Estruturas de control

Coma noutras linguaxes, é fundamental coñecer as estruturas de control de fluxo de datos de *Python* para case calquera programa que desexemos crear. Recordamos que en estruturas de control incluíamos as estruturas **secuenciais**, as **condicionais** e as **repetitivas.**

\+ -

» **Estruturas Secuenciais**

Manuel Torres Búa. *Secuencia* (CC BY-SA)

Unha estrutura secuencial, como sabemos, é aquela na que o fluxo de información do programa realízase en etapas consecutivas. Non nos pararemos neste tipo xa que podemos afirmar que inclúe calquera posibilidade de programación con *Python* sempre que non se altere sucesión ordenada do fluxo de información

» **Estruturas condicionais**

» **Estruturas repetitivas**

As estruturas repetitivas son aquelas nas que hai presenza de *bucles* ou *iteracións*. O programa vai repetir un ou varios bloques de código, ben un número determinado de veces, ben en función do cumprimento ou non dunha condición.

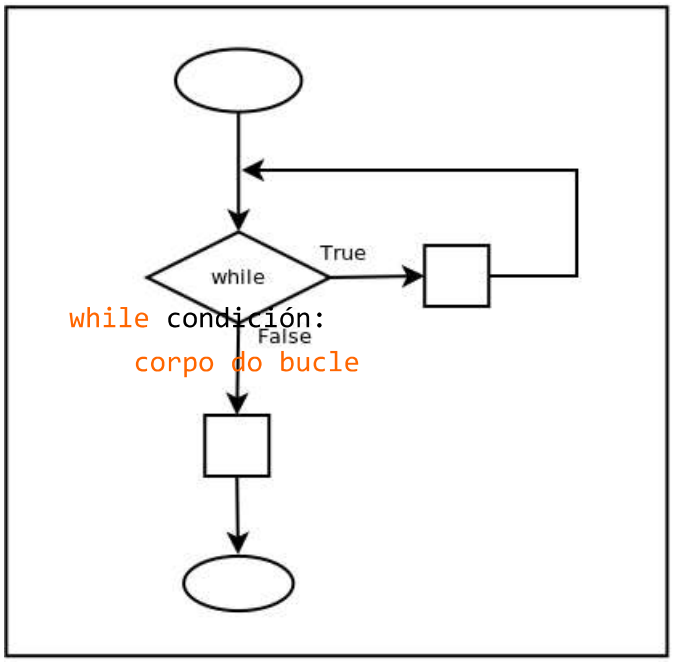
for...in

while

Fundación Plone y amigos. *While* (GNU/GPL)

Un Bucle *while* permítenos repetir a execución dun grupo de instrucións mentres se cumpre unha

condición (é dicir: sempre que a condición teña o valor *True*). A sintaxe do ciclo *while* é a seguinte:



Cando chega a un bucle *while*, *Python* avalía a condición e, se é verdadeira, executa o corpo do ciclo. Unha vez que se executa o corpo do ciclo, o proceso repítese valorando a condición novamente e, se é verdadeira, o corpo do ciclo execútase; unha e outra vez. Só cando a condición é falsa, o corpo do ciclo non se executará e continuará a execución do resto do programa.

Exemplo:

```
numero = int(input("Escribe un número maior que dez: "))
while numero <= 10:
    print("Escribiu un número que non é maior que 10! Inté
    numero = int(input("Escribe un número maior que dez: "
print("Grazas pola súa colaboración")

======== RESTART: C:/Users/proba.py ==========
Escribe un número maior que dez: 8
Escribiu un número que non é maior que 10! Inténteo de nov
Escribe un número maior que dez: 9
Escribiu un número que non é maior que 10! Inténteo de nov
Escribe un número maior que dez: 10
Escribiu un número que non é maior que 10! Inténteo de nov
Escribe un número maior que dez: 58
Grazas pola súa colaboración
>>>
```

Rudimentos de Python

Algúns exemplos

Presentamos a continuación algúns exemplos de programas desenvolvidos en *Python,* amosando o seu código fonte.

Exemplo 1

Programa para calcular o índice de masa corporal (IMC), a partir da introdución dos datos peso e altura.

Cálculo do IMC

```
print("CÁLCULO DO ÍNDICE DE MASA CORPORAL (IMC)")
peso = float(input("Canto pesa? "))
altura = float(input("Canto mide en metros? "))

imc = peso / altura**2

print(f"O seu IMC é {round(imc, 1)}")
print("Un IMC moi alto indica obesidade. Os valores \"normais\" de IM
print("entre 20 y 25, pero estes límites dependen da idade, do sexo,
print("constitución física, etc.")
```

```
========= RESTART: C:/Usersproba.py ==========

CÁLCULO DO ÍNDICE DE MASA CORPORAL (IMC)
Canto pesa? 75
Canto mide en metros? 1.76
O seu IMC é 24.2
Un IMC moi alto indica obesidade. Os valores "normais" de IMC están
entre 20 y 25, pero estes límites dependen da idade, do sexo, da
constitución física, etc.
```

Exemplo 2 (if...elif...else)

Programa que solicite os coeficientes dunha ecuación de segundo grao (a x^2 + bx + c = 0) e calcule e escriba a solución.

Unha ecuación de segundo grao ou non ten solución, ou ten unha solución única, ou ten dúas solucións ou ben todos os números son unha solución. A fórmula das solucións cando hai dúas solucións é

$x = (-b \pm \sqrt{(b^2 - 4ac)}) / (2a)$

Algúns exemplos de posibles respostas

a	b	c	Solución
1	-2	2	Sen solución real
2	-7	3	Dúas solucións: 0,5 e 3,0
1	2	1	Unha solución: -1.0
0	0	5	Sen solución
0	0	0	Todos os números son solución
0	3	2	Unha solución: -0.666 ...

Programa

Esta é unha posible solución

```
print("ECUACIÓN A X² + B X + C = 0")
a = float(input("Escriba o valor do coeficiente a: "))
b = float(input("Escriba o valor do coeficiente b: "))
c = float(input("Escriba o valor do coeficiente b: "))

d = b*b - 4*a*c
if a == b == c == 0:
    print("Todos os números son solución")
elif a == b == 0:
    print("Sen solución")
```

```
    elif a == 0:
         print(f"Unha solución: {- c / b}")
    elif d < 0:
        print("Sen solución real")
    elif d == 0:
        print(f"Unha solución: {- b / (2*a)}")
    else:
        print(f"Dúas solucións: {(-b - d**0.5) / (2*a)} y "
              f"{(-b + d**0.5) / (2*a)}")


========== RESTART: C:/Users/proba.py ==========
ECUACIÓN A X² + B X + C = 0
Escriba o valor do coeficiente a: -8
Escriba o valor do coeficiente b: 6
Escriba o valor o coeficiente b: 7
Dúas solucións: 1.3827822185373186 y -0.6327822185373186
```

Exemplo 3 (range())

Programa que solicita tres números enteiros, calcula e escribe a lista de múltiplos do terceiro número entre os dous primeiros (incluídos os mesmos se son múltiplos do número indicado)

Programa

```
print("MÚLTIPLOS ENTRE VALORES")
inicial = int(input("Escriba o número enteiro inicial: "))
final = int(input("Escriba o número enteiro final: "))

if final < inicial:
    print("¡O número final debe ser maior que o inicial!")
else:
    paso = int(input("De que número quere os múltiplos?: "))
    if paso <= 0:
        print("¡Os múltiplos deben ser dun número enteiro maior que c
    else:
        if inicial % paso != 0:
            inicial2 = inicial // paso * paso + paso
        else:
            inicial2 = inicial
```

```
            print(f"Entre {inicial} e {final} hai "
                  f"{len(range(inicial2, final + 1, paso))} múltiplos de 
            print(list(range(inicial2, final + 1, paso)))
```

```
========== RESTART: C:/Users/proba.py ==========
MÚLTIPLOS ENTRE VALORES
Escriba o número enteiro inicial: 15
Escriba o número enteiro final: 12
¡O número final debe ser maior que o inicial!
>>>
========== RESTART: C:/Users/proba.py ==========
MÚLTIPLOS ENTRE VALORES
Escriba o número enteiro inicial: 15
Escriba o número enteiro final: 42
De que número quere os múltiplos?: -3
¡Os múltiplos deben ser dun número enteiro maior que cero!
>>>
========== RESTART: C:/Users/proba.py ==========
MÚLTIPLOS ENTRE VALORES
Escriba o número enteiro inicial: 15
Escriba o número enteiro final: 42
De que número quere os múltiplos?: 3
Entre 15 e 42 hai 10 múltiplos de 3
[15, 18, 21, 24, 27, 30, 33, 36, 39, 42]
>>>
```

Exemplo 4 (for...in)

Programa que pide un número enteiro maior que 0 e que calcula e escribe os seus divisores

Programa

```
print("DIVISORES")
numero = int(input("Escriba un número enteiro maior que cero: "))

if numero <= 0:
    print("Pedinlle un número entero maior que cero!")
else:
    print(f"Os divisores do {numero} son ", end="")
    for i in range(1, numero + 1):
        if numero % i == 0:
            print(i, end=" ")
```

```
========== RESTART: C:/Users/proba.py ==========
DIVISORES
Escriba un número enteiro maior que cero: -3
Pedinlle un número entero maior que cero!
>>>
========== RESTART: C:/Users/proba.py ==========
DIVISORES
Escriba un número enteiro maior que cero: 123
Os divisores do 123 son 1 3 41 123
>>>
```

(*for...in* con contadores)

Programa que pide un número enteiro maior que 1 e que calcula e escribe se é un número primo ou non

Programa

```
    print("NÚMERO PRIMO")
    numero = int(input("Escriba un número enteiro maior que 1: "))

    if numero <= 1:
        print("Pedinlle un número enteiro maior que 1!")
    else:
        contador = 0
        for i in range(1, numero + 1):
            if numero % i == 0:
                contador = contador + 1
        if contador == 2:
            print(f"{numero} é primo")
        else:
            print(f"{numero} non é primo")


========== RESTART: C:/Users/proba.py ==========
NÚMERO PRIMO
Escriba un número enteiro maior que 1: -3
Pedinlle un número enteiro maior que 1!
>>>
========== RESTART: C:/Users/proba.py ==========
NÚMERO PRIMO
Escriba un número enteiro maior que 1: 8
8 non é primo
>>>
========== RESTART: C:/Users/proba.py ==========
NÚMERO PRIMO
Escriba un número enteiro maior que 1: 5
5 é primo
>>>
```

(*for...in* con acumuladores)

Programa que pregunta cantos números se queren introducir, e que calcula e escribe o máximo, o mínimo e a media aritmética.

Programa

```
    print("MAIOR, MENOR E MEDIA ARITMÉTICA")
    numero = int(input("Cantos valores vai introducir? "))

    if numero <= 0:
        print("Imposible!")
    else:
        valor = float(input("Escriba o número 1: "))
        minimo = maximo = suma = valor
        for i in range(2, numero + 1):
            valor = float(input(f"Escriba o número {i}: "))
            suma = suma + valor
            if valor < minimo:
                minimo = valor
            if valor > maximo:
                maximo = valor
        print(f"O número máis pequeno dos introducidos é {minimo}")
        print(f"O número máis grande dos introducidos é {maximo}")
        print(f"A media dos números introducidos é {suma / numero}")


========== RESTART: C:/Users/proba.py ==========
MAIOR, MENOR E MEDIA ARITMÉTICA
Cantos valores vai introducir? 6
Escriba o número 1: 25
Escriba o número 2: 47
Escriba o número 3: 268
Escriba o número 4: 2565
Escriba o número 5: 55
Escriba o número 6: 45
O número máis pequeno dos introducidos é 25.0
O número máis grande dos introducidos é 2565.0
A media dos números introducidos é 500.8333333333333
>>>
```

Exemplo 5 (while)

Programa que solicita un valor límite positivo e que vaia solicitando números ata que a súa suma supere o límite inicial.

Programa

```
limite = float(input("Escriba o valor límite: "))
while limite <= 0:
    limite = float(input("O límite debe ser maior que 0. Inténteo de 

numero = float(input("Escriba un número: "))
suma = 0
suma += numero

while suma < limite:
    numero = float(input("Escriba outro número: "))
    suma += numero

print()
print(f"Xa superou o límite. A suma dos números positivos introducido
print("Programa rematado")
```

```
========== RESTART: C:/Users/proba.py ==========
Escriba o valor límite: -254
O límite debe ser maior que 0. Inténteo de novo: 254
Escriba un número: 2
Escriba outro número: 65
Escriba outro número: 21
Escriba outro número: 84
Escriba outro número: 35
Escriba outro número: 2
Escriba outro número: 69

Xa superou o límite. A suma dos números positivos introducidos é 278.0.
Programa rematado
>>>
```

Estes exemplos son unha tradución e adaptación dos contidos en *Introducción a la programación con Python* de Bartolomé Sintes Marco (Mclibre) con licenza CC BY SA.

Licenciado baixo a Licenza Creative Commons Recoñecemento Compartir igual 4.0