Seguridad Informática

Programación con Python

Virgilio Castro Rendón
Fernando Marcos Parra Arroyo

Temario

- Fundamentos de programación
  - Asignación y comparación
  - Funciones y valores de retorno
  - Ciclos y condiciones
- Convenciones léxicas y sintaxis básica
  - Bloques declarados e implícitos
  - Comentarios
  - Convenciones de nombres
  - PEP
- Estructuras de datos
  - Listas
  - Tuplas y secuencias
  - Diccionarios
- Estructura de código
  - Funciones
  - Clases
  - Módulos
  - Alcance, Herencia, Extensión
- Buenas practices
  - Documentación
  - Manejo de errors
  - CWE Top 25
- Técnicas y módulos
  - Breve revision de la biblioteca estándar
  - Módulos externos de interés

Detalles del curso

Inicio: 26 de noviembre

Fin: 5 de diciembre (examen)

Horario: 15:00 – 18:00

Descanso de 20 minutos

Repositorio de github:

virgilio.castro.rendon@gmail.com

Evaluación:

- Participación: 10%
- Ejercicios de clase: 20%
- Tareas: 40 %
- Examen: 30%

Programación en Python

Historia

- Creado a principios de la década de los 90s por Guido van Rossum mientras trabajaba en el CWI (Holanda)
- Sucesor del lenguaje ABC
- Nombre inspirado en el grupo Monthy Python
- En el año 2000 cambia sus políticas para poder ser compatible con licencias GPL, con lo que comienza a ser ampliamente aceptado en la comunidad de software libre
- Guido actualmente sigue teniendo un papel fundamental en el desarrollo del lenguaje (Benevolent Dictator for Life)

Diferentes clasificaciones de los lenguaje

- Compilado o interpretado
- Paradigmas
  - Estructurado
  - Orientado a objetos
  - Funcional
- Tipado
  - Dinámico (Comprobación de tipos en tiempo de ejecución)
  - Estático (Comprobación de tipos en tiempo de compilación)

Python

- Tipado dinámico
- Lenguaje interpretado
- Con soporte para los 3 paradigmas mencionados

Zen de Python

```
>>> import this
The Zen of Python, by Tim Peters

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!
```

Intérprete de Python

- Ejecutar un módulo:
  - python –m SimpleHTTPServer

```
CURSO-PYTHON> python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

- Ejecutar un comando:
  - python –c ‘print bin(123)’

```
CURSO-PYTHON> python -c 'print bin(123)'
0b1111011
CURSO-PYTHON>
```

Intérprete de Python

- Ejecutar un script:
  - python script.py

```
CURSO-PYTHON> python test.py
Ingresa un número:      10
Ingresa otro número:    40
Suma:   50
```

- Abrir Shell interactiva:
  - python

```
CURSO-PYTHON> python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'Estoy reprobado'
Estoy reprobado
>>>
```

Hola Mundo

- Algo un poco complejo

```
CURSO-PYTHON> cat hola_mundo.py
print 'Hola Mundo'
CURSO-PYTHON> python hola_mundo.py
Hola Mundo
```

- Descargar: hola_mundo.py
  - Shebang
  - Tipo de codificación
  - Comentarios
  - Definición de funciones
  - Docstrings

Tipos de datos

- Los tipos de datos básicos en python son:
  - Números enteros
    - Base decimal, binaria, hexadecimal, etc.
  - Números reales
    - Notación con punto decimal y notación científica
  - Números complejos
  - Booleanos
    - True, False
  - Cadenas
  - None
- Una variable puede almacenar cualquier tipo de dato en cualquier momento de la ejecución

Descargar: tipos.py

Operaciones básicas

- En general no es posible hacer operaciones entre tipos de datos distintos.
- Excepciones:
  - Entero con flotante
  - Cadena con entero (multiplicación)
- Si se desea operar entre tipos distintos, se debe hacer un ‘casteo’ explícitamente

Operaciones básicas

- Introducción de datos por el usuario
  - num1 = int(raw_input('Introduce un número: '))

- **Suma:**
  - n = 5 + 3
  - 8
- **Resta:**
  - n = 5 – 3
  - 2
- **Multiplicación:**
  - n = 5 * 3
  - 15
- **División:**
  - n = 5 / 3
  - 1
  - n = 5.0 / 3
  - 1.6666
- **Módulo:**
  - n = 5 % 3
  - 2
- **Exponente:**
  - n = 5 ** 3
  - 125
- **Concatenación:**
  - n = 'Hola' + ' Mundo'
  - 'Hola Mundo'
- **Multiplicación de cadenas**:
  - n = 'Hola' * 2
  - 'HolaHola'

Variables

- Una variable es una localidad de memoria que almacena un valor. Es identificada a través de un nombre.
- El nombre de una variable puede estar formado por minúsculas, mayúsculas, números y guion bajo (_)
- Una variable puede iniciar solamente con minúsculas, mayúsculas o guion bajo (_).

Asignación de variables

- En Python, una variable puede almacenar diferentes tipos de datos durante la ejecución.

- Se pueden asignar múltiples variables en una línea

```
var1, var2, var3 = 'hola', 'mundo', '!!'
print var1,var2,var3
var1, var2, var3 = 5, 6, 7
print var1,var2,var3
```

```
CURSO-PYTHON> python var.py
hola mundo !!
5 6 7
CURSO-PYTHON>
```

Asignación de variables

- Existen diversos operadores de asignación, siendo el principal y más usado el símbolo “=“.

Operador	Equivalencia
a = 5	Asigna un valor a la variable ‘a’
a += 5	a = a + 5
a -= 5	a = a - 5
a *= 5	a = a * 5
a /= 5	a = a / 5
a %= 5	a = a % 5
a **= 5	a = a ** 5

Listas

- Estructura de datos que permite almacenar múltiples valores, los cuales son accesibles mediante un índice.
- Los datos contenidos en una lista pueden ser de cualquier tipo.
- Se declaran usando corchetes.
- Las listas en Python permiten acceder a los datos de una forma avanzada (slices).
- Existen diversas funciones que son aplicables a las listas.

Listas

- lista1 = [1, 2, 3, 4, 5]
- lista2 = [‘elemento1’, ’elemento2’, 3, 4.0, [‘a’,’b’]]

- Operaciones:
  - Slices, suma, multiplicación
- Funciones:
  - len, max, min
- Métodos
  - append, count, index, insert, pop, remove, reverse, sort

Listas - slices

- lista1 = [‘Juan’, ‘Pedro’,’Jose’,’Fernando’]

Juan	Pedro	Jose	Fernando
0	1	2	3
-4	-3	-2	-1

- lista1[inicio:final:salto]
- lista1[0] -> ‘Juan’
- lista1[:2] -> [‘Juan’,’Pedro’]
- lista1[2:] -> [‘Jose’,’Fernando’]
- lista1[-1] -> ‘Fernando’

Descargar: slices.py

Listas – operaciones y funciones

Python provee diversas formas de manipular listas, algunas de ellas afectarán a la lista original y otras generarán listas nuevas, sin afectar la original.

- lista1 = [1,2,3,4]
- lista2 = [5,6,7,8,9]

lista1*2 -> [1,2,3,4,1,2,3,4]

lista1 + lista2 -> [1,2,3,4,5,6,7,8,9]

len(lista1) -> 4

max(lista2) -> 9

min(lista2) -> 5

Descargar: listas.py

Estructuras de control

- Permiten la implementación de los algoritmos en el paradigma de programación estructurado.
- Cada una de las estructuras de control implica la creación de un bloque de código.
- Ciclos:
  - while
  - for
- Condicionales:
  - if
  - elif
  - else

While

- Repite un conjunto de acciones hasta que la condición evaluada sea falsa.

```
var1 = 0
while var1 < 10:
        print '\t%s' % var1
        var1 += 1
```

```
CURSO-PYTHON> python while.py
        0
        1
        2
        3
        4
        5
        6
        7
        8
        9
CURSO-PYTHON>
```

For

- A diferencia del ciclor “for” de otros lenguajes de programación, en python implica iterar sobre un conjunto de datos como una lista. Ejecuta un bloque de instrucciones por cada elemento de la lista.

```
lista1 = [‘Pedro’,’Jose’,’Juan’,’Diego’]
for nombre in lista1:
        print nombre + ‘ no es becario’
```

```
CURSO-PYTHON> python for.py
Pedro no es becario
Jose no es becario
Juan no es becario
Diego no es becario
CURSO-PYTHON>
```

For

- Si se quiere emular un comportamiento como el que tiene es ciclo “for” en otros lenguajes de programación, se usa la función “range” que genera rangos definidos de números enteros.
- Tiene un comportamiento similar a los slices

```
for i in range(1,15,3)
        print ‘%s -> %s’ % (i, i**2)
```

```
CURSO-PYTHON> python for.py
1 -> 1
4 -> 16
7 -> 49
10 -> 100
13 -> 169
```

If-else

- Ejecuta el primer bloque de instrucciones si la condición es verdadera y el segundo si la condición es falsa.

```
var1 = 10
if var1 < 5:
        print '%s es menor a 5' % var1
else:
        print '%s es mayor a 5' % var1
```

```
CURSO-PYTHON> python if.py
10 es mayor a 5
```

If-elif-else

- Evalúa una condición, si es falsa evalúa la siguiente y así sucesivamente. Ejecuta el bloque de instrucciones correspondiente a la condición verdadera.

```
var1 = int(input('Introduce un numero: '))
if var1 < 5:
        print '%s es menor a 5' % var1
elif var1 == 5:
        print '%s es igual a 5' % var1
else:
        print '%s es mayor a 5' % var1
```

```
CURSO-PYTHON> python elif.py
Introduce un numero: 5
5 es igual a 5
```

Break

- Instrucción que, si se usa, debe estar dentro de un ciclo
- Es una instrucción que provoca que el ciclo se rompa y se termine por completo.

```
for i in range(1,2000):
        print i
        if i == 5:
                break
```

```
CURSO-PYTHON> python break.py
1
2
3
4
5
CURSO-PYTHON>
```

Continue

- Instrucción que, si se usa, debe estar dentro de un ciclo
- Es una instrucción que la iteración actual del ciclo termine y se ejecute la siguiente.

```
for i in range(1,10):
        if i % 2 == 0:
                continue
        print i
```

```
CURSO-PYTHON> python continue.py
1
3
5
7
9
CURSO-PYTHON>
```

Operadores de comparación

- Estos operadores provocan un resultado booleano (True o False), por lo tanto son idóneos para ser usados en un bloque if, elif o while.

Operador	Efecto
a == b	Si los dos valores son iguales, es verdadero
a != b	Si los dos valores son diferentes, es verdadero
a > b	Si a es mayor que b, es verdadero
a >= b	Si a es mayor o igual que b, es verdadero
a < b	Si a es menor que b, es verdadero
a <= b	Si a es menor o igual que b, es verdadero
a in [a,b,c]	Si a se encuentra en el arreglo, es verdadero
a not in [a,b,c]	Si a no se encuentra en el arreglo, es verdadero
‘abc’ in ‘abcdef’	Si la cadena ‘abc’ se encuentra en la cadena ‘abcdef’, es verdadero
‘abc’ not in ‘abcdef’	Si la cadena ‘abc’ no se encuentra en la cadena ‘abcdef’, es verdadero

Funciones

- Son bloques de código reutilizables mediante un nombre.
- Las funciones en Python por defecto regresan el valor nulo (None).
- Se pueden especificar diversos valores de retorno.
- Una función puede, o no, recibir argumentos.
- Se pueden especificar valores por defecto en los argumentos.

Descargar: funciones.py

Ejercicio de clase 1

- Descargar el archivo e1.py para que.:
  - Valide a los becarios sin importar si están en mayúsculas o minúsculas
  - Inserte a los becarios aprobados en mayúsculas
  - Ordene la lista de aprobados después de hacer una inserción

- Agregar una función que permita borrar a un becario aprobado a partir de su nombre completo.
  - Recibe nombre del becario a eliminar
  - Regresa verdadero (True) si se pudo eliminar y falso (False) si no se encontraba en la lista.

Ejercicio de clase2

- Hacer una función que valide si una cadena es un palíndromo.
  - Recibe: cadena a validar
  - Regresa: booleano (True / False)

Tarea 1

- Hacer una función que encuentre el palíndromo más grande contenido en una cadena
  - Recibe: cadena a validar
  - Regresa: cadena (palíndromo más grande)
- Hacer función que determine si un número es primo o no
  - Recibe: entero (número a validar)
  - Regresa: booleano
- Hacer función RECURSIVA que agregue en una lista los primeros n números primos
  - Regresa: lista

Tarea 2

- Función RECURSIVA que genere contraseñas seguras
  - Regresa: cadena (contraseña)
  - Rand.py