



Facultad: Ingeniería
Escuela: Ingeniería en Computación
Asignatura: Autómatas y Compiladores

Guía Introductoria

Introducción

Este manual ha sido elaborado para orientar al estudiante de Autómatas y Compiladores en el desarrollo de sus prácticas de laboratorio. El uso de este manual debe ser antes, durante y después de la práctica, de tal forma que ofrece un método facilitador en su proceso de enseñanza/aprendizaje durante esta asignatura.

En el desarrollo de esta asignatura se ha designado realizar las prácticas en 16 sesiones semanales de laboratorios, los que incluyen 11 prácticas y tres desafíos prácticos, durante los cuales, el estudiante aplicará los conceptos y las técnicas fundamentales necesarias para la implementación de una herramienta a la solución de problemas. Todas las guías de laboratorio están estructuradas de la siguiente forma:

- Introducción.
- Objetivos específicos.
- Materiales y equipo a utilizar.
- Procedimiento.
- Análisis de resultados.
- Investigación y ejercicios complementarios.
- Fuentes de Consulta.

Objetivos Específicos

- Conocer los lineamientos de las prácticas de laboratorio.
- Conocer los lenguajes de programación que se utilizarán en las prácticas.

Material y Equipo

- Guía de laboratorio N° 1.
- Computadora con Python 3.0 o superior.
- Memoria USB.

Sobre las prácticas

El contenido de esta asignatura está orientado a estudiar las diferentes herramientas de formalización de las expresiones en el contexto de la teoría de autómatas y compiladores

Se utilizarán diferentes herramientas para los fines que se buscan. Entre las herramientas a utilizar, tenemos:

- IDLE Python
- Spyder with Anaconda (IDE) entorno de trabajo de Python.
- Jflap (simulador de autómatas)
- Java (para la generación de analizadores léxicos)

- Bodhi Linux

Introducción Teórica

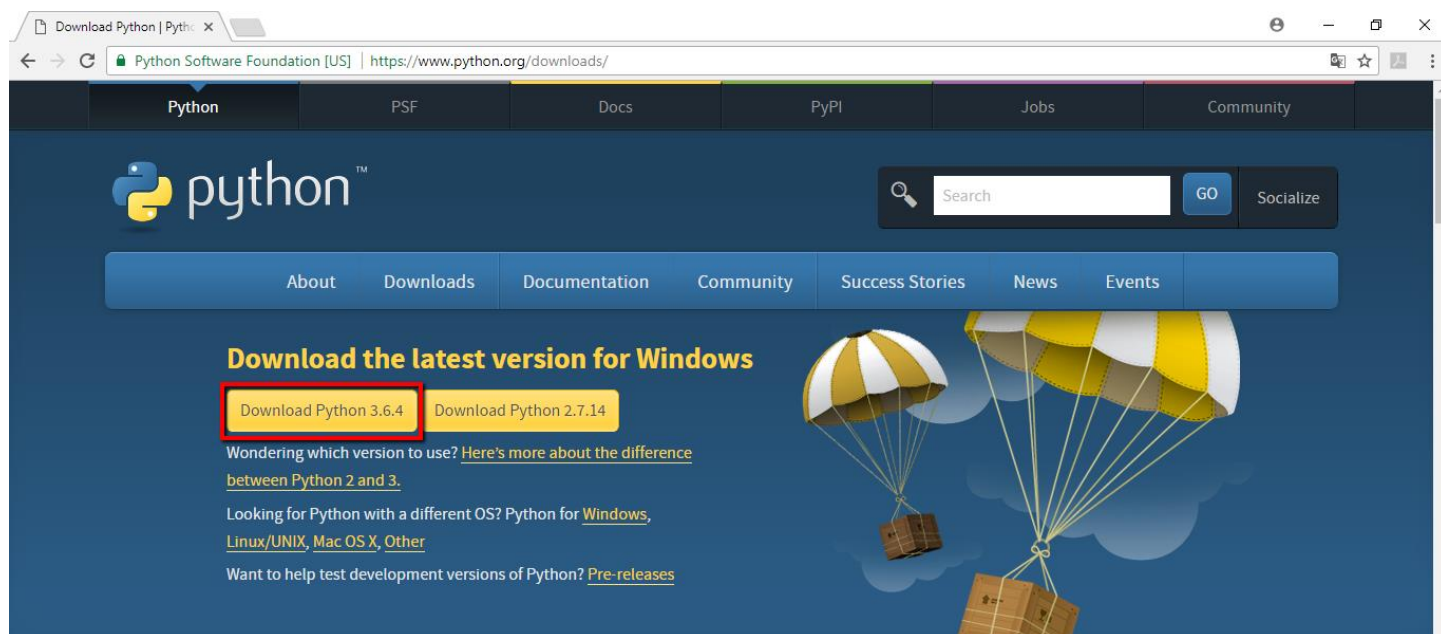
Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel, así como también con un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada hacen de este lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y en código fuente para las principales plataformas desde el sitio de Python, <https://www.python.org/>, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, además, contiene documentación adicional.

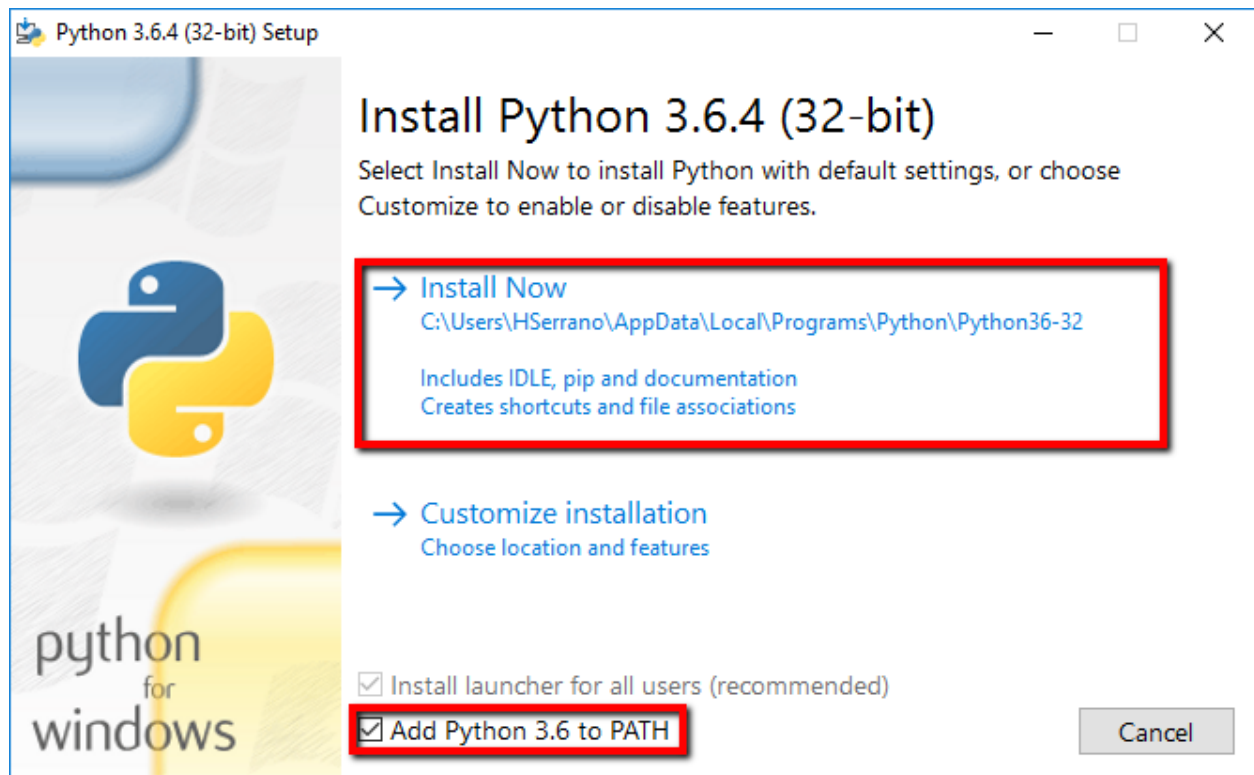
El intérprete de Python puede extenderse fácilmente con nuevas funcionalidades y tipos de datos implementados en C o C++ (u otros lenguajes accesibles desde C). Python también puede usarse como un lenguaje de extensiones para aplicaciones personalizadas.

Procedimiento

1. Vamos a comenzar por acceder al sitio oficial de Python para poder descargar una de las versiones con la cual trabajaremos en las prácticas de laboratorio. La versión que vamos a descargar es la última (3.6.4).



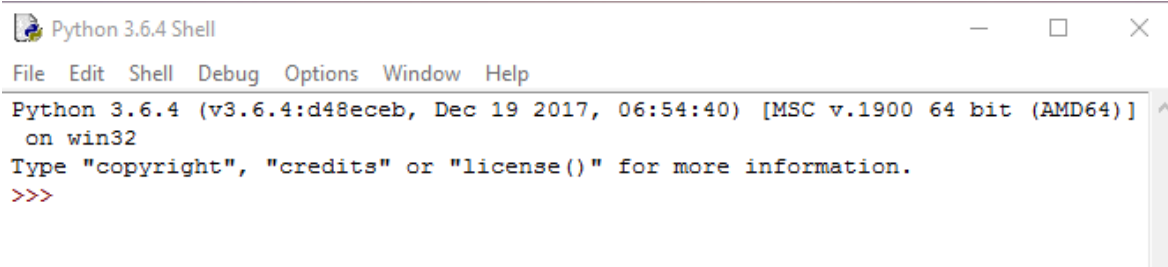
2. Procedemos a la instalación.



3. Una vez finalizado el proceso de instalación, vamos a verificar desde la consola de comandos la versión de Python:
 NOTA: La opción "Add Python 3.6 to PATH" se selecciona para que se cree automáticamente una variable de entorno y pueda ser reconocido en la consola.
 Simplemente digite el comando **"python"**.

4. Abrimos la aplicación y tendremos acceso a la siguiente interfaz (Modo Interactivo):

Se dice que estamos usando el intérprete en modo interactivo, cuando los comandos son leídos desde una terminal. En este modo espera el siguiente comando con el prompt primario, usualmente tres signos mayor-que (>>>); para las líneas de continuación espera con el prompt secundario, por defecto tres puntos (...), o bien muestra espacios en blanco. Antes de mostrar el prompt primario, el intérprete muestra un mensaje de bienvenida reportando su número de versión y una nota de copyright:



Inicialmente, se utilizará esta área de trabajo.

Las líneas de continuación son necesarias cuando queremos ingresar un constructor multilínea. Como en el ejemplo, veamos la sentencia if (explicada con más detalle más adelante).

```
>>> Dragon_Ball_Super = True
>>> if Dragon_Ball_Super:
    print("Vamos por el siguiente capítulo!!!!")
```

```
Vamos por el siguiente capítulo!!!!
```

El intérprete de Python y su entorno

Codificación del código fuente

Por default, los archivos fuente son tratados como codificados en UTF-8. En esa codificación, los caracteres de la mayoría de los lenguajes del mundo pueden ser usados simultáneamente en literales, identificadores y comentarios, a pesar de que la biblioteca estándar usa solamente caracteres ASCII para los identificadores, una convención que debería seguir cualquier código que sea portable. Para mostrar estos caracteres correctamente, tu editor debe reconocer que el archivo está en UTF-8 y usar una tipografía que soporte todos los caracteres del archivo.

Para especificar una codificación distinta de la por defecto, una línea de comentario especial debe ser agregada como la primera línea del archivo. La sintaxis es como sigue:

```
# -*- coding: encoding -*-
```

Donde encoding es uno de los codecs válidos soportados por Python.

Por ejemplo, para indicar que el encoding Windows-1252 es el usado, la primera línea de tu código fuente debe ser:

```
# -*- coding: cp-1252 -*-
```

Una introducción formal a Python

En los siguientes ejemplos, las entradas y salidas son distinguidas por la presencia o ausencia de los prompts (>>> y ... o simplemente en lugar los dos puntos habrá espacios en blanco). Para reproducir los ejemplos debes escribir todo lo que está después del prompt, cuando éste aparezca, las líneas que no comiencen con el prompt son las salidas del intérprete.

Los comentarios en Python comienzan con el carácter numeral en caso de que sean de una sola línea. En caso de que sean de varias líneas puede incluir los comentarios dentro de un bloque de tres comillas simples, así:

```
""" este es el comentario de múltiples líneas """
```

Nunca uses las comillas (' o ") con los tipos de datos booleanos y siempre se debe utilizar una inicial mayúscula. Python distingue entre mayúsculas y minúsculas.

Usando Python como una calculadora

Vamos a probar algunos comandos simples de Python. El intérprete actúa como una calculadora simple, se puede ingresar una expresión y éste escribirá dos valores. La sintaxis es sencilla: los operadores `+`, `-`, `*`, `/` funcionan como en la mayoría de los lenguajes (por ejemplo, Pascal o C); los paréntesis `(())` pueden ser usados para agrupar. Por ejemplo:

```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8/5
1.6
>>> 8/5 # La división siempre devolverá un número de punto flotante
1.6
>>>
```

Los números enteros (por ejemplo, 2, 4, 20) son de tipo `int`, aquellos con una parte fraccional (por ejemplo, 5.0, 1.6) son de tipo `float`.

La división (`/`) siempre retorna un punto flotante. Para hacer *floor división* y obtener un resultado entero (descartando cualquier resultado fraccional) se puede usar el operador `//`; para calcular el resto se puede usar `%`.

```
>>> 17 / 3 # La división clásica retorna un punto flotante
5.666666666666667
>>> 17 // 3 # La división entera descarta la parte funcional
5
>>> 17 % 3
2
>>> 17 % 3 # El operador % retorna el resto de la división
2
>>> 5 * 3 + 2 # Resultado * Divisor + Resto
17
>>> |
```

Con Python, es posible utilizar el operador `**` para calcular potencias.

```
>>> 5**2 # 5 al cuadrado
25
>>> 2**7 # 2 a la potencia de 7
128
>>>
```

El signo igual (`=`) es usado para asignar un valor a una variable. Luego, ningún resultado es mostrado antes del próximo prompt:

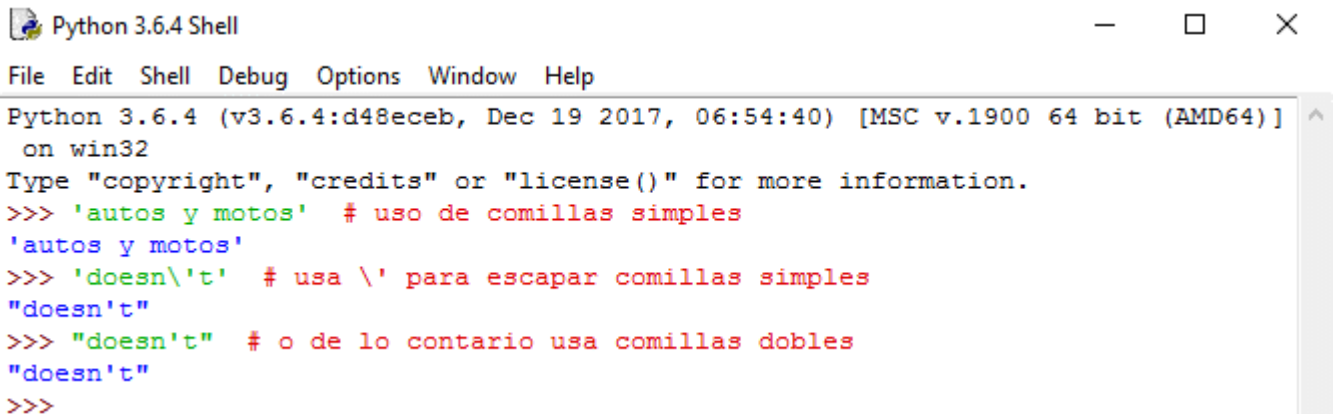
```
>>> ancho = 20
>>> largo = 5*9
>>> ancho * largo
900
>>> |
```

En el modo interactivo, la última expresión impresa es asignada a la variable `_`. Esto significa que cuando se esté usando Python como una calculadora de escritorio, es más fácil seguir calculando, por ejemplo:

```
>>> Impuesto=12.5/100
>>> Precio=100.50
>>> Precio*Impuesto
12.5625
>>> Precio+ _
113.0625
>>> round(_,2)
113.06
>>>
```

Cadenas de caracteres

Además de números, Python puede manipular cadenas de texto, las cuales pueden ser expresadas de distintas formas. Pueden estar encerradas en comillas simples (`'...'`) o dobles (`"..."`) con el mismo resultado. `\` puede utilizarse para escapar comillas:



```
Python 3.6.4 Shell
File Edit Shell Debug Options Window Help
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:54:40) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> 'autos y motos' # uso de comillas simples
'autos y motos'
>>> 'doesn\'t' # usa \' para escapar comillas simples
"doesn't"
>>> "doesn't" # o de lo contrario usa comillas dobles
"doesn't"
>>>
```

Las cadenas de texto se pueden indexar (subíndices), el primer carácter de la cadena tiene el índice 0. No hay un tipo de dato para los caracteres, un carácter es simplemente una cadena de longitud 1:

```
>>> palabra = 'Python'
>>> palabra[0] # obtenemos caracter en la posición 0
'P'
>>> palabra[5] # obtenemos caracter en la posición 5
'n'
>>>
```

Nota que `-0` es lo mismo que 0, los índices negativos comienzan de `-1`.

Además de los índices, las rebanadas también están soportadas. Mientras que los índices son usados para obtener caracteres individuales, las rebanadas permiten obtener sub-cadenas:

```
>>> palabra[:2] # caracteres desde la posición 0 (incluida) hasta la 2 (excluida)
'Py'
>>> palabra[2:5] # caracteres desde la posición 2 (incluida) hasta la posición 5 (excluida)
'tho'
>>>
```

Nota como el primer valor siempre es incluido, y que el último es siempre excluido. Esto asegura que `s[:i] + s[i:]` siempre sea igual a `s`:

```
>>> palabra[:2] + palabra[2:]
'Python'
>>> palabra[:4] + palabra[4:]
'Python'
>>>
```

Los índices de las rebanadas tienen valores por defecto útiles, el valor por defecto del primer índice es cero, el valor por defecto del segundo índice es la longitud de la cadena a rebanar.

```
>>> palabra[:2] # caracter desde el principio hasta el caracter 2 (excluido)
'Py'
>>> palabra[4:] # caracter desde el principio hasta el caracter 4 (incluido) hasta el final
'on'
>>> palabra[-2:] # caracter desde la ante-penúltima (incluida) hasta el final
'on'
```

- La función incorporada ***len()*** devuelve la longitud de una cadena de texto:

```
>>> Cadena = 'supercalifrastrilisticoespialidoso'
>>> len(Cadena)
33
```

- La función ***lower()*** es para trabajar solo con minúsculas en una cadena. Se puede llamar a dicho método de la siguiente forma:
`"Pablo".lower()`
- La función ***upper()*** permite trabajar toda la cadena en mayúsculas. Se puede llamar a dicho método de la siguiente forma:
`"Pablo".upper()`
- Ahora, veamos el método ***str()***, que es menos directo. El método `str()` convierte una cadena en algo que no es una cadena. Por ejemplo:
`str(2)`
Debe convertir 2 en "2"

Ejercicios propuestos

A continuación, se presentan los siguientes ejercicios, el instructor de la práctica revisará cada resultado.

1. Escriba un comentario de una sola línea, debe asegurarse de que comience con #.
2. Declarar una variable **monty** y haga que sea igual a **True**.
3. Declare una variable llamada **Python** y haga que sea igual a **1.234**.
4. Declara u una tercera variable **monty_python** y haga que sea igual a **Python** al cuadrado.

Investigación Complementaria

1. Realizar una investigación sobre:
 - a. Tipos integrados: f-string y formatstring
2. Presentar un ejemplo de cada tipo integrado.
3. Descargar entorno de trabajo Spyder with Anaconda: <https://www.spyder-ide.org/>

Bibliografía

- Python, Guido Van Rossum