Capacitación Python
Laboratorio 0

Nelson R. Salinas

Abril 3, 2018

En esta primera sesión se instalarán el software necesario para el desarrollo de la capacitación y se reconocerán los principales tipos de datos.

1. Instalación de Anaconda

Anaconda es un sistema de distribución de software desarrollado en Python, principalmente librerías. Dado que el desarrollo de software usualmente implica la creación de dependencia a otro software, es bastante popular el uso de sistemas de distribución de paquetes que resuelven el árbol de dependencias por el usuario. Anaconda es bastante eficiente al respecto y su énfasis ha sido la distribución de librerías empleadas en ciencia de datos. Otro de los beneficios de usar Anaconda es que la instalación contiene el intérprete de Python y varias librerías básicas.

Existen cuatro versiones de Anaconda dependiendo de la versión de Python subyacente (2.7 o 3.6) y la cantidad de librarías instaladas por defecto: casi ninguna (Miniconda) o todas las básicas para cualquier proyecto en ciencia de datos (Anaconda). En esta ocasión se descargará e instalará la versión para Python 2.7. Si aún no tiene instalado dicho software, descarge el paquete deseado (Miniconda o Anaconda) e instálelo, para lo cual puede consultar la página de ayuda oficial.

2. Ejecución interactiva

La forma más sencilla de utilizar Python y ejecutar operaciones sencillas es a través de algunas aplicaciones que permiten ejecutar las órdenes (casi) línea por línea. La más popular es la consola de Python.

2.1. Consola básica

La consola interactiva de Python siempre se distribuye con la instalación de Python básica.

1. Navege en el menu de programas hasta la suite Anaconda/Miniconda.

2. Inicialice la terminal a través de un comando llamado simplemente “python”.

3. Digite los comandos del Código 1.

4. Si por alguna razón la instalación de Anaconda/Miniconda no permite el acceso a Python desde la terminal, pase al numeral 2.2.

Código 1: Operaciones básicas.

```
1 # Estimacion del area de un circulo
2 diam = 4
3 area = (diam / 2.0) ** 2 * 3.14159
4 area # Cual es el area resultante?
5
6 # Estimacion del volumen de un cilindro
7 altura = 13.4
8
9 # Que operacion ejecutaria para obtener el volumen?
```

2.2. ipython

Una segunda opción es la consola ipython. Una de las ventajas de esta consola respecto a la anterior es que proporciona resaltado sintáctico, un historial de comandos ejecutados, autocompletado de comandos y otras funciones de utilidad.

1. Navegue el menu de programas de su sistema operativo hasta llegar al conjunto de programas Anaconda/Miniconda.

2. Ejecute el comando `ipython`.

3. Digite los comandos del Código 1.

Funciones útiles de `ipython`:

- <inicio de comando>, Tab. Permite autocompletar comandos.
- <inicio de comando>, CTRL + P. Navega a través de los comandos anteriormente ejecutados en order inverso.
- <inicio de comando>, CTRL + N. Navega a través de los comandos anteriormente ejecutados en order inverso.
- CTRL + R, <parte de comando>. Permite buscar comandos previamente ejecutados ingresando cualquier segmento del mismo.
- `%rerun` + número de comando. Ejecuta de nuevo el comando de una celda predeterminada.

2.3. Jupyter notebook

El problema de usar las consolas es que es necesario digitar todo el código cada vez que se vaya a utilizar, lo cual aumenta la probabilidad de cometer errores. Afortunadamente existen herramientas interactivas que permiten ejecutar el código en bloques y guardarlo en archivos. Una de dichas herramientas es el "cuaderno" (notebook) de Jupyter, bastante popular entre el gremio de analistas de datos. Al ejecutar un cuaderno de Jupyter Python, éste es cargado en un servidor local que puede ser accedido a través del explorador de Internet instalado en la computadora. Estos cuadernos interactivos también proporcionan los mismos utilidades adicionales de ipython: resaltado sintáctico, autocompletado, etc.

1. Navegue el menu de programas de su sistema operativo hasta llegar al conjunto de programas Anaconda/Miniconda.
2. Ejecute el comando `jupyter notebook`. A continuación una nueva ventana en el explorador de internet será cargada.
3. Para comenzar un nuevo cuaderno oprima "New" → "Python 2".

   El cuaderno está dividido en celdas, y las celdas puede ser de texto o código. Una vez se escribe código en la celda, éste se ejecuta presionando "play" o CTRL + Enter. La información contenida en un cuaderno puede ser guardada en un archivo usando la opción "File" → "Save and checkpoint" o simplemente presionando CTRL + s.
4. En una nueva celda de código digite los comandos del Código 1 y ejecútelos.

3. Ejecución de scripts

La ejecución interactiva de código es sumamente útil durante el desarrollo, pero una vez la pieza de código ha alcanzado estabilidad es más sencillo ejecutar todos los comandos de una sola vez. Esta tarea se realiza direccionando todos los comandos de Python desde un archivo fuente hacia el intérprete como un proceso en el "background" de la línea de comando.

1. En un editor de texto digite el siguiente código y guárdelo como el archivo "test.py":

Código 2: Comandos del archivo "test.py".

```
1 diam = 4
2 area = (diam / 2.0) ** 2 * 3.14159
3 altura = 13.4
4 volumen = area * altura
5 print "Volumen:",volumen
```

2. Ahora, desde la linea de comandos, navegue hasta la ubicación del archivo "test.py".
3. Ejecute el comando `python test.py`

4. Tipos de datos

Existen 8 tipos de datos fundamentales en Python 2.7. Cada tipo tiene la capacidad de interactuar de una manera específica con las funciones básicas, a veces simplemente no interactuando. A continuación se presentan una serie de expresiones que se pueden realizar en la consola interactiva (consola básica o ipython) para familiarizarse con las capacidades de cada uno de estos datos.

4.1. Números enteros (`int`)

Existen dos tipos de datos numéricos ampliamente utilizados: enteros (`int`) y reales (`float`). Ambos proveen soporte de operaciones aritméticas tradicionales; sin embargo, se debe tener en cuenta que cualquier operación entre dos enteros *siempre* produce un entero. Por otro lado, la operación módulo (operador `%`) retorna el residuo de una división.

Código 3: Operaciones básicas con `int`.

```
1  myint = 15
2  print myint
3  myint + 100
4  myint * 2
5  myint / 3
6  myint / 4
7  myint % 3
8  myint % 4
9  myint ** 2
10 myint += 1
11 print myint
```

4.2. Números reales (`float`)

Código 4: Operaciones básicas con `float`.

```
1  myfloat = 15.5
2  print myfloat
3  myfloat * 2
4  myfloat / 2
5  myfloat % 3
6  myfloat % 4
7  round(myfloat)
8  myfloat = float(myint)
```

4.3. Booleanos (`bool`)

Los operadores booleanos son simplemente verdadero (`True`) o falso (`False`). Los números 1 y 0 también los pueden reemplazar en todas las operaciones, respectivamente. Son bastante empleados para realizar pruebas condicionales.

Código 5: Operaciones básicas con `bool`.

```
1  mybool = True
2  mybool == False
3  mybool == True
4  mybool == 0
5  mybool == 1
6
7  if mybool:
8      print "Existo!"
9  else:
10     print "Oximoron!"
```

4.4. Nulo (`None`)

El tipo Nulo es empleado usualmente para inicializar variables y realizar pruebas acerca de su existencia.

Código 6: Operaciones básicas con `None`.

```
1  nada = None
2
3  if nada:
4      print "Oximoron!"
5  else:
6      print "Existo!"
7
8  nada = 120
9
10 if nada:
11     print "Oximoron!"
12 else:
13     print "Existo!"
```

4.5. Listas (`list`)

Las listas son los tipos no-escalares por excelencia en Python, y pueden estar compuestos por objetos de cualquier tipo de dato. Dado que son una serie de objetos cuyo orden no cambia, son muy empleados para 1) agrupar elementos relacionados y 2) llevar registro del orden de una operación. Algunas de la operaciones más importantes son acceso a los elementos (individualmente o en grupos), adición de elementos (individualmente o en grupo), eliminación de elementos y comprobación de la existencia de elementos en la lista.

Código 7: Operaciones básicas con `list`.

```
1  mylist = [0, 1, 2.0, False, None]
2  mylist = range(10)
3  print mylist
4  mylist = range(100,110)
```

```
5  len(mylist)
6  mylist[:3]
7  mylist[8:]
8  mylist[4:6]
9  mylist[-1]
10 mylist.append(110)
11 mylist = mylist + [111, 112]
12 print mylist
13 mylist.pop()
14 print mylist
15 mylist.pop(5)
16 print mylist
17 mylist.index(110)
```

4.6. Tuplas (`tuple`)

Las tuplas son bastante similares a las listas, la principal diferencia es que son inmutables: su estructura ni su contenido pueden cambiar una vez han sido inicializadas. Esta es la razón por la cual sólo comparten con las listas los métodos de declaración y acceso. Dada su estabilidad, son los objetos preferidos para retornar no escalares de funciones.

Código 8: Operaciones básicas con `tuple`.

```
1 mytuple = (1, 2, 10, 20)
2 mylist = list(mytuple)
3 mytuple[:2]
4 mylist[1] = 3
5 print mylist
6 mytuple[1] = 3
```

4.7. Texto (`str` y `unicode`)

El texto puede ser declarado como objetos `str` o `unicode`, las cuales son maneras diferentes de representar el texto en memoria. Ambos se comportan como listas, especialmente respecto a su acceso e indexación. Tienen además varias funciones que facilitan la ejecución de ciertas tareas usuales en estructuras de texto, como lo son las conversiones entre minúsculas y mayúsculas.

Código 9: Operaciones básicas con `str`.

```
1 mystring = "Hola mundo!"
2 print mystring
3 len(mystring)
4 mystring[:3]
5 mystring[8:]
6 mystring[4:6]
7 mystring[-1]
8 mystring += " Adios mundo!"
9 print mystring
```

```
10 mystring.index('a')
11 mystring.upper()
12 mystring.lower()
13 mystring.title()
14 mystring.split(' ')
15 mystring.split('')
16 str(myint)
17 str(myfloat)
```