Facultad de Ciencias de la Administración

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

PROGRAMACIÓN I

Unidad III – Tipos de Datos

Tipos de datos elementales: Segunda parte.

Tecnicatura Universitaria en Desarrollo Web

Facultad de Ciencias de la Administración

Universidad Nacional de Entre Ríos

- **Objetivos**
  - Comprender cómo se definen variables, contantes se realizan operaciones con ellas.
  - Entender el concepto de tipo de datos.
  - Identificar las diferencias entre los tipos de datos elementales.
  - Inspeccionar cómo se representa en memoria cada tipo.
- **Temas a desarrollar:**
  - Variables y constantes. Definición. Alcance.
  - Expresiones. Operadores lógicos. Operadores aritméticos. Operadores de cadenas de texto. Precedencia.
  - Tipos de datos elementales: booleano, entero, punto flotante, cadena de texto.
  - Conversión de tipos.
  - Representación de datos en memoria.

Strings o Cadenas de caracteres

- Los ***strings*** no son como los enteros, números de punto flotante o booleanos.
- *Un **string** es una **secuencia**, es decir, una colección ordenada de valores. En este caso de **caracteres**.*
- Un ***carácter*** es la unidad más pequeña en un sistema de escritura. Incluye letras, dígitos, símbolos, puntuación e incluso espacios en blanco o directivas como saltos de línea.
- A diferencia de otros lenguajes, los strings en **Python** son inmutables. Es decir, no puede cambiar una cadena. Nos queda entonces, copiar partes de strings a otro string para obtener el mismo efecto.
- Alternativas para la creación de cadenas:
  - » `'Snap'` → **‘Snap’** `# Comillas simples`
  - » `"Crackle"` → **‘Crackle’** `# Comillas dobles`
  - » `'''Boom!'''` → **‘Boom!’** `# Triple comillas simples`
  - » `"""Quich!"""`→ **‘Quich!’** `# Triple comillas dobles`
  - » **`str(4.5)`** → **‘4.5’** `# Conversión de valores a string`
- El propósito principal de las múltiples alternativas para crear strings es que los mismos puedan contener comillas simples o dobles dentro del texto:
  - » `"Calle O' Connor"` → **“Calle O’ Connor”**
  - » `'"Hoy puede ser un gran día!"'` → **‘“Hoy puede ser un gran día!”’**

Strings o Cadenas de caracteres (2)

- Los strings con triple comillas son útiles para crear strings multilínea.
  - » `"""Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam placerat, sem ut luctus consectetur, ex elit cursus orci, eget tempus purus arcu vel risus."""`
  - » `'Lorem ipsum dolor sit amet, consectetur adipiscing elit.\n Nullam placerat, sem ut luctus consectetur, ex elit cursus orci, eget tempus purus arcu vel risus.'`
- La función **`print()`** remueve las comillas de los strings e imprime su contenido.
  - » **`print(“We”, ‘will’, “rock”, ‘you’)`** → **`We will rock you`**
- Existe el ***string vacío***. Que si bien no tiene ningún carácter es completamente válido.
  - » `''` → `''`
  - » `""` → `''`
  - » `''''''` → `''`

Secuencias de escape

- Supongamos que en un string queremos escribir un **“enter”** o **salto de línea**. ¿Cómo hacemos? ¿Y si quiero hacer tabulaciones?
- **Python** nos permite expresar caracteres que tienen un significado especial (y que de otra forma serían muy difíciles de expresar) utilizando ***secuencias de escape***.
- Expresamos una **secuencia de escape** con el carácter **** (**barra invertida** o **back slash**).
- Podemos hacer saltos de línea o “enters” con **\n** dentro del string**:**
  - `» november_rain ="Nothin' lasts forever\nAnd we both know hearts can change\nAnd it's hard to hold a candle\nIn the cold November rain"`
  - `» print(november_rain)`
  - `Nothin' lasts forever`
  - `And we both know hearts can change`
  - `And it's hard to hold a candle`
  - `In the cold November rain`
- Las tabulaciones se hacen con **\t**
  - `» print("Sweet Child\t O' mine")`
  - `Sweet Child	O’ mine`

Indexación utilizando [ ]

- Para obtener un único carácter de un string debemos, justo después el nombre del string, y entre corchetes debemos especificar su **posición** o **índice**.
  - La primer posición (más a la izquierda) accesible es 0, la siguiente 1 y así sucesivamente.
  - La última posición (más a la derecha) puede ser accedida usando -1, yendo más a la izquierda -2, -3 y así sucesivamente.
  - Los índices válidos de un string de longitud **x**, son enteros desde **-x** hasta **+x-1**.
  - **» letras = 'abcdefghijklmnopqrstuvwxyz'**
  - **» letras[0]** → **'a'**
  - **» letras[1]** → **'b'**
  - **» letras[-1]** → **'z'**
  - **» letras[-2]** → **'y'**
  - **» letras[5]** → **'f'**
- Si especificamos un índice mayor a la longitud del string. Tendremos un error:
  - **» letras[100]**
  - **Traceback (most recent call last):**
  - **IndexError: string index out of range**

Substrings con slice

- Un ***substring*** es el nombre con el que se conoce a una parte de un **string**.
- Podemos extraer un **substring** de un string utilizando un ***slice*** (del inglés rebanada).
- Definimos un **slice** usando corchetes, una posición de **`inicio`**, una de **`fin`** y opcionalmente un **`paso`** entre ellos.
  - Se pueden omitir algunos de los valores.
  - El **slice** incluirá los caracteres coincidentes con **`inicio`** y el **`fin`** menos uno.
- **`[:]`** extrae la secuencia completa desde el principio a final.
- **`[inicio:]`** especifica desde **`inicio`** hasta el final.
- **`[:fin]`** especifica desde el principio al **`final`** menos uno.
- **`[inicio:fin]`** indica desde el principio hasta el **`fin`** menos 1
- **`[inicio:fin:paso]`** extrae desde **`inicio`** hasta el **`fin`** menos 1, omitiendo la cantidad de caracteres especificadas en **`paso`**.

Substrings con slice (2)

- Si el ***paso es positivo*** los desplazamientos se incrementan desde inicio hacia la derecha siendo el primero 0, luego 1, y así sucesivamente y -1, -2 y así sucesivamente desde el final hacia la izquierda en caso de tratarse de un ***paso negativo***.
- **Ejemplos:**
  - » **letras = ‘abcdefghijklmnopqrstuvwxyz’**
  - » **letras[:]** → **‘abcdefghijklmnopqrstuvwxyz’** # el string completo.
  - » **letras[20:]** → **'uvwxyz'** # desde 20 hasta el final.
  - » **letras[12:15]** → **'mno'** # desde 12 hasta 14.
  - » **letras[-3:]** → **'xyz'** # los últimos 3 caracteres.
  - » **letras [::7]** → **‘ahov’** # de principio a fin salteando de a 7 caracteres.
  - » **letras[::-1]** → **'zyxwvutsrqponmlkjihgfedcba'** # empieza al final y con paso negativo llega al principio.

Longitud

- Hasta ahora hemos utilizado algunos caracteres de puntuación especiales para tratar con strings como el operador **+**.
- **Python** tiene muchas ***funciones integradas*** (porciones de código con nombre) que pueden llevar a cabo determinadas operaciones.
- La función **`len()`** (del inglés length o longitud) cuenta la cantidad de caracteres en un string.
  - `» len(letras)`
  - `26`
  - `» vacio = ''`
  - `» len(vacio)`
  - `0`

Bibliografía

- Óscar Ramírez Jiménez: ***"Python a fondo"*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Bill Lubanovic. ***"Introducing Python"***. 2Da Edición. O' Reilly. 2020.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Zed A. Shaw: ***"Learn Python 3 the Hard Way"***. 1era Edición. Ed. Addison-Wesley. 2017.