

Introducción a Python

Programación en Python

Proyecto de Inteligencia Artificial Interdisciplinaria y Escuela de Ciencia y Tecnología, UNSAM

1er cuatrimestre 2020

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquéense y vuelvan a preguntar**. ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquéense y vuelvan a preguntar**. ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquése y vuelvan a preguntar**. ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquése y vuelvan a preguntar**. ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquése y vuelvan a preguntar**. ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquéense y vuelvan a preguntar.** ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquéense y vuelvan a preguntar.** ¡Así funciona esto!

Objetivos

- Buscamos profundizar en la idea que computadora es una **herramienta** que brinda inmensas posibilidades.
- Vamos a **resolver** problemas usando la computadora.
- Aprender a encarar problemas, plantearlos y resolverlos, en el medio vamos a tener que **programar**.
- Y vamos a **programar** bien:
 - usando estructuras de datos adecuadas,
 - evitando complejidad excesiva y
 - documentando lo que hacemos.

El cuerpo docente está para ayudar: **pregunten, experimenten, equivoquéense y vuelvan a preguntar**. ¡Así funciona esto!

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python
- Un poco de teoría de algoritmos

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python
- Un poco de teoría de algoritmos
- Ejemplos interesantes y tareas desafiantes

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python
- Un poco de teoría de algoritmos
- Ejemplos interesantes y tareas desafiantes
- Tiene entregas semanales

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python
- Un poco de teoría de algoritmos
- Ejemplos interesantes y tareas desafiantes
- Tiene entregas semanales
- Cada uno debe tipear y enviar su solución por correo.

El curso

- Es un curso intenso, de 16 semanas, con 2 parciales individuales.
- Incluye detalles técnicos del lenguaje python
- Un poco de teoría de algoritmos
- Ejemplos interesantes y tareas desafiantes
- Tiene entregas semanales
- Cada uno debe tipear y enviar su solución por correo.
- Cada uno debe evaluar su entrega.

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático
- manejo de errores en python

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático
- manejo de errores en python
- clases (programación orientada a objetos), herencia, generadores

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático
- manejo de errores en python
- clases (programación orientada a objetos), herencia, generadores
- complejidad de algoritmos, técnicas de divide and conquer.

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático
- manejo de errores en python
- clases (programación orientada a objetos), herencia, generadores
- complejidad de algoritmos, técnicas de divide and conquer.
- algoritmos de búsqueda, ordenamiento

El curso

Los temas que proponemos para el curso son:

- estructuras de control en python
- estructuras de datos en python
- concepto de recursión
- entrada y salida de datos
- visualización de datos
- intro a aprendizaje automático
- manejo de errores en python
- clases (programación orientada a objetos), herencia, generadores
- complejidad de algoritmos, técnicas de divide and conquer.
- algoritmos de búsqueda, ordenamiento
- algoritmos estocásticos: técnica de Montecarlo.

El curso

- Pregunten todo lo que no entiendan. Interrumpan.

El curso

- Pregunten todo lo que no entiendan. Interrumpan.
- Ustedes son muchos y nosotros sólo dos.

El curso

- Pregunten todo lo que no entiendan. Interrumpan.
- Ustedes son muchos y nosotros sólo dos.
- Dénsese soporte entre ustedes.

El curso

- Pregunten todo lo que no entiendan. Interrumpan.
- Ustedes son muchos y nosotros sólo dos.
- Dénsese soporte entre ustedes.
- Haremos grupo de whatsapp

Las entregas

Los ejercicios se dividen en tres categorías:

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
 - 2 optativos (*)
 - 3 desafíos (**)
- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
- Si les queda tiempo lean e intenten los optativos y,

Las entregas

Los ejercicios se dividen en tres categorías:

- ❶ obligatorios (que esperamos que hagan, completos)
 - ❷ optativos (*)
 - ❸ desafíos (**)
- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
 - Si les queda tiempo lean e intenten los optativos y,
 - Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
 - 2 optativos (*)
 - 3 desafíos (**)
- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
 - Si les queda tiempo lean e intenten los optativos y,
 - Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)
 - Son unas 4hs de cursada semanal y otras horas de práctica en casa.

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
- Si les queda tiempo lean e intenten los optativos y,
- Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)
- Son unas 4hs de cursada semanal y otras horas de práctica en casa.
- Quizas en 2-3hs extras lo puedan resolver. Quizas a veces se queden 10hs extras.

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
- Si les queda tiempo lean e intenten los optativos y,
- Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)
- Son unas 4hs de cursada semanal y otras horas de práctica en casa.
- Quizas en 2-3hs extras lo puedan resolver. Quizas a veces se queden 10hs extras.
- Recuerden! Se pueden discutir en grupo y consultar con los compañeros pero cada uno debe tipear y comprender su solución.

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
- Si les queda tiempo lean e intenten los optativos y,
- Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)
- Son unas 4hs de cursada semanal y otras horas de práctica en casa.
- Quizas en 2-3hs extras lo puedan resolver. Quizas a veces se queden 10hs extras.
- Recuerden! Se pueden discutir en grupo y consultar con los compañeros pero cada uno debe tipear y comprender su solución. Y enviarla por correo.

Las entregas

Los ejercicios se dividen en tres categorías:

- 1 obligatorios (que esperamos que hagan, completos)
- 2 optativos (*)
- 3 desafíos (**)

- Los obligatorios deben enviarlos por correo ANTES de la siguiente clase.
- Si les queda tiempo lean e intenten los optativos y,
- Si les tiente hacer alguno de los desafíos que son más largos o más sutiles haganlo! (esperamos poder discutirlos también)
- Son unas 4hs de cursada semanal y otras horas de práctica en casa.
- Quizas en 2-3hs extras lo puedan resolver. Quizas a veces se queden 10hs extras.
- Recuerden! Se pueden discutir en grupo y consultar con los compañeros pero cada uno debe tipear y comprender su solución. Y enviarla por correo. Y completar el formulario de autoevaluación.

Python

¿Por qué Python?

Python

Python es :

- Open Source

Python

Python es :

- Open Source
- Cross-platform

Python

Python es :

- Open Source
- Cross-platform
- Tiene un fuerte desarrollo en una gran variedad de áreas:
 - Desarrollo de aplicaciones con interface gráfica (jueguitos por ejemplo).
 - Análisis de datos (ahora *Ciencia de Datos*)
 - Simulación numérica.
 - Desarrollo para Web e Internet.
 - Robótica.

Python

Python es :

- Open Source
- Cross-platform
- Tiene un fuerte desarrollo en una gran variedad de áreas:
 - Desarrollo de aplicaciones con interface gráfica (jueguitos por ejemplo).
 - Análisis de datos (ahora *Ciencia de Datos*)
 - Simulación numérica.
 - Desarrollo para Web e Internet.
 - Robótica.

*Es un lenguaje que se usa cada día más.
Aprender `Python` **es** una inversión segura.*

Pero... Python, ¿Estás seguro?

Most in-demand programming languages of 2019

Based on Indeed.com job postings in the USA - Feb 1, 2019

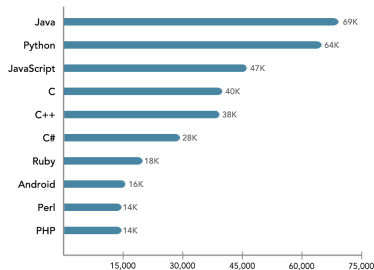
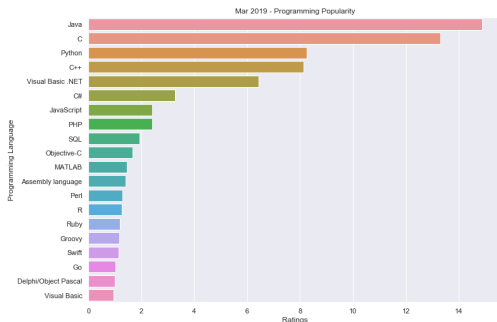


Image Source: CodingNomads



Pero... Python, ¿Estás seguro?

Most in-demand programming languages of 2019

Based on Indeed.com job postings in the USA - Feb 1, 2019

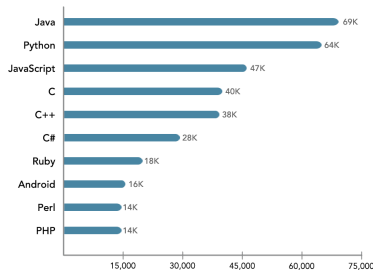
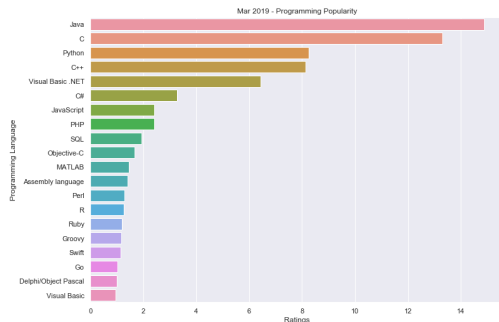


Image Source: CodingNomads



Sí, estamos seguros

No importa el ranking o el sitio que se mire, Python siempre está dentro de los primeros lugares de interés o crecimiento año a año.

Python como Calculadora

```
>>> print("Hola Mundo!")  
Hola Mundo!
```

Python como Calculadora

```
>>> print("Hola Mundo!")  
Hola Mundo!
```

```
>>> n=10  
>>> print(n)  
10  
>>> type(n)  
<class 'int'>  
>>> n+n  
20  
>>> n*n  
100  
>>> n**3  
1000  
>>> 123+321  
444  
>>>
```

Python como Calculadora

```
>>> print("Hola Mundo!")  
Hola Mundo!
```

```
>>> n=10  
>>> print(n)  
10  
>>> type(n)  
<class 'int'>  
>>> n+n  
20  
>>> n*n  
100  
>>> n**3  
1000  
>>> 123+321  
444  
>>>
```

```
>>> a=0.1  
>>> type(a)  
<class 'float'>  
>>>
```

Python como Calculadora

```
>>> print("Hola Mundo!")  
Hola Mundo!
```

```
>>> n=10  
>>> print(n)  
10  
>>> type(n)  
<class 'int'>  
>>> n+n  
20  
>>> n*n  
100  
>>> n**3  
1000  
>>> 123+321  
444  
>>>
```

```
>>> a=0.1  
>>> type(a)  
<class 'float'>  
>>>
```

```
>>> a==0.1  
True  
>>>  
>>> type(a==0.1)  
<class 'bool'>
```

Python como Calculadora

```
>>> print("Hola Mundo!")
Hola Mundo!
```

```
>>> n=10
>>> print(n)
10
>>> type(n)
<class 'int'>
>>> n+n
20
>>> n*n
100
>>> n**3
1000
>>> 123+321
444
>>>
```

```
>>> a=0.1
>>> type(a)
<class 'float'>
```

```
>>> a==0.1
True
>>>
>>> type(a==0.1)
<class 'bool'>
```

```
>>> a+a+a+a+a+a+a+a+a+a
0.9999999999999999
>>> a+a+a+a+a+a+a+a+a+a==1.0
False
```

Python como Calculadora

```
>>> print("Hola Mundo!")
Hola Mundo!
```

```
>>> n=10
>>> print(n)
10
>>> type(n)
<class 'int'>
>>> n+n
20
>>> n*n
100
>>> n**3
1000
>>> 123+321
444
>>>
```

```
>>> a=0.1
>>> type(a)
<class 'float'>
```

```
>>> a==0.1
True
>>>
>>> type(a==0.1)
<class 'bool'>
```

```
>>> a+a+a+a+a+a+a+a+a+a
0.9999999999999999
>>> a+a+a+a+a+a+a+a+a+a==1.0
False
```

```
>>> b=12
>>> h=6
>>> (b*h)/2
36.0
```


Python como Calculadora

```
>>> print("Hola Mundo!")
Hola Mundo!
```

```
>>> n=10
>>> print(n)
10
>>> type(n)
<class 'int'>
>>> n+n
20
>>> n*n
100
>>> n**3
1000
>>> 123+321
444
>>>
```

```
>>> a=0.1
>>> type(a)
<class 'float'>
```

```
>>> a==0.1
True
>>>
>>> type(a==0.1)
<class 'bool'>
```

```
>>> a+a+a+a+a+a+a+a+a+a
0.9999999999999999
>>> a+a+a+a+a+a+a+a+a+a==1.0
False
```

```
>>> b=12
>>> h=6
>>> (b*h)/2
36.0
```

```
>>> (b*h)//2
36
>>> 3//2
1
>>> □
```

Python como Calculadora

```
>>> print("Hola Mundo!")
Hola Mundo!
```

```
>>> n=10
>>> print(n)
10
>>> type(n)
<class 'int'>
>>> n+n
20
>>> n*n
100
>>> n**3
1000
>>> 123+321
444
>>>
```

```
>>> a=0.1
>>> type(a)
<class 'float'>
```

```
>>> a==0.1
True
>>>
>>> type(a==0.1)
<class 'bool'>
```

```
>>> a+a+a+a+a+a+a+a+a+a
0.9999999999999999
>>> a+a+a+a+a+a+a+a+a+a==1.0
False
```

```
>>> b=12
>>> h=6
>>> (b*h)/2
36.0
```

```
>>> (b*h)//2
36
>>> 3//2
1
>>>
```

```
>>> 3%2
1
>>> 185%100
85
```

Python como Calculadora

```
>>> print("Hola Mundo!")
Hola Mundo!
```

```
>>> n=10
>>> print(n)
10
>>> type(n)
<class 'int'>
>>> n+n
20
>>> n*n
100
>>> n**3
1000
>>> 123+321
444
>>>
```

```
>>> a=0.1
>>> type(a)
<class 'float'>
```

```
>>> a==0.1
True
>>>
>>> type(a==0.1)
<class 'bool'>
```

```
>>> a+a+a+a+a+a+a+a+a+a
0.9999999999999999
>>> a+a+a+a+a+a+a+a+a+a==1.0
False
```

```
>>> b=12
>>> h=6
>>> (b*h)/2
36.0
```

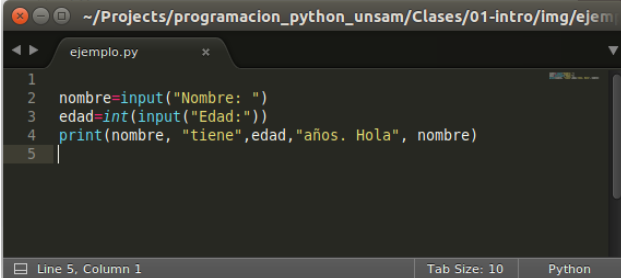
```
>>> (b*h)//2
36
>>> 3//2
1
>>> □
```

```
>>> 3%2
1
>>> 185%100
85
```

```
>>> (1+2*3)**2
49
```

Scripts y Python en modo script

Scripts y ejecución desde línea de comandos

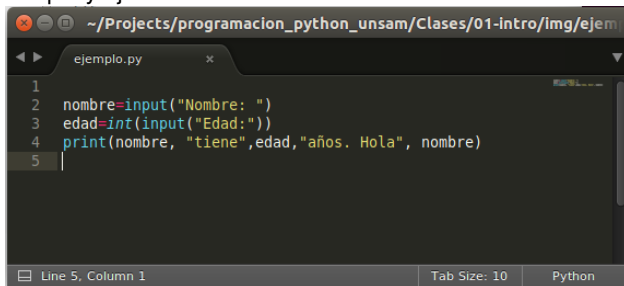
A screenshot of a code editor window. The title bar shows the file path: ~/Projects/programacion_python_unsam/Clases/01-intro/img/ejemplo.py. The editor has a tab labeled 'ejemplo.py'. The code is as follows:

```
1
2 nombre=input("Nombre: ")
3 edad=int(input("Edad:"))
4 print(nombre, "tiene",edad,"años. Hola", nombre)
5
```

The status bar at the bottom indicates 'Line 5, Column 1', 'Tab Size: 10', and 'Python'.

Scripts y Python en modo script

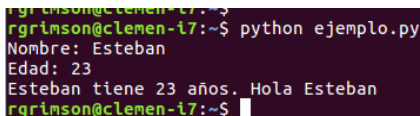
Scripts y ejecución desde línea de comandos



A screenshot of a code editor window titled "ejemplo.py". The window shows a Python script with the following code:

```
1
2 nombre=input("Nombre: ")
3 edad=int(input("Edad:"))
4 print(nombre, "tiene",edad,"años. Hola", nombre)
5
```

The status bar at the bottom indicates "Line 5, Column 1", "Tab Size: 10", and "Python".



A screenshot of a terminal window showing the execution of the Python script. The prompt is "rgrimson@clemen-i7:~\$". The user enters "python ejemplo.py". The output is:

```
rgrimson@clemen-i7:~$ python ejemplo.py
Nombre: Esteban
Edad: 23
Esteban tiene 23 años. Hola Esteban
rgrimson@clemen-i7:~$
```

Python como Calculadora

Ejercicio

¿Cuántos segundos hay en 42 minutos 42 segundos?

Ejercicio

¿Cuántos kilómetros son 6 millas? (un kilómetro corresponde a 0,6214 millas)

Ejercicio

Si corre una carrera de 6 millas en 42 minutos 42 segundos, ¿cuál fue tu velocidad promedio en km/h?

Ejercicio

Escriba un script que pregunte el radio r de una esfera y calcule su volúmen, $\frac{4}{3}\pi r^3$. Ejecute el script desde la línea de comandos para responder ¿cuál es el volumen de una esfera de radio 6?

Spyder - IDE

~/Projects/PMW_Tychonov/Scripts - Spyder (Python 3.6)

Editor - /home/rgrimson/.config/spyder-py3/temp.py

```

1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4 This is a temporary script file.
5 """
6
7
8 r = 6
9
10 pi = 3.141592
11
12 vol = 3/4*pi*r**3
13

```

Name	Type	Size	Value
pi	float	1	3.141592
r	int	1	6
vol	float	1	508.93790400000006

Variable ex... File ex... Find i... Help Static code an...

IPython console

Console 8/A

```

Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 19:07:31)
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: r = 6

In [2]: pi = 3.141592

In [3]: vol = 3/4*pi*r**3

```

Otros tipos de datos: Listas

- En Python existen las **listas**, que sirven para almacenar múltiples valores:

```
mi_lista = []
```

En esta línea, asignamos a la variable `mi_lista` una lista vacía (no contiene ningún elemento).

Otros tipos de datos: Listas

- En Python existen las **listas**, que sirven para almacenar múltiples valores:

```
mi_lista = []
```

En esta línea, asignamos a la variable `mi_lista` una lista vacía (no contiene ningún elemento).

- Veamos ahora cómo hacer para agregar valores en esta lista:

```
mi_lista = []
```

```
mi_lista.append(4)  
mi_lista.append(8)  
mi_lista.append(1.0)
```

```
len(mi_lista)
```

Otros tipos de datos: Listas

- En Python existen las **listas**, que sirven para almacenar múltiples valores:

```
mi_lista = []
```

En esta línea, asignamos a la variable `mi_lista` una lista vacía (no contiene ningún elemento).

- Veamos ahora cómo hacer para agregar valores en esta lista:

```
mi_lista = []
```

```
mi_lista.append(4)  
mi_lista.append(8)  
mi_lista.append(1.0)
```

```
len(mi_lista)
```

- Si ejecutamos este fragmento de código observamos lo siguiente:

Spyder: Ejemplo de una lista

The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script named `temp.py` with the following code:

```

1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8 mi_lista = []
9
10 mi_lista.append(4)
11 mi_lista.append(8)
12 mi_lista.append(1.0)
13
14 len(mi_lista)
15 |

```

The Variable explorer on the right shows the variable `mi_lista` of type `list` with a size of 3 and a value of `[4, 8, 1.0]`.

The IPython console at the bottom shows the execution of the script:

```

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: mi_lista = []

In [2]: mi_lista.append(4)

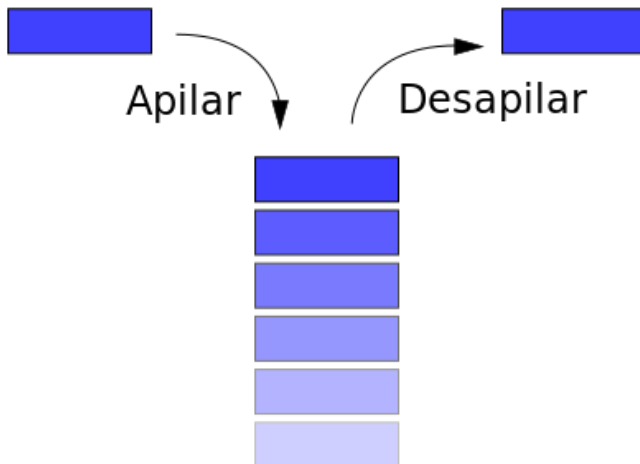
In [3]: mi_lista.append(8)

In [4]: mi_lista.append(1.0)

In [5]: len(mi_lista)
Out[5]: 3

```

Pilas - LIFO (append y pop)



Pilas : append y pop

Editor - /home/rgrimson/Projects/PMW_Tychonov/Scripts/untitled0.py

temp.py X ej1.py X untitled0.py* X

```

1 #!/usr/bin/env python3
2 #-*- coding: utf-8 -*-
3 """
4 Created on Mon Mar  9 17:04:23 2020
5
6 @author: rgrimson
7 """
8
9 mi_lista = []
10
11 mi_lista.append(4)
12 mi_lista.append(8)
13 mi_lista.append(12)
14
15 mi_lista
16
17 mi_lista.pop()
18
19 mi_lista
20

```

Variable explorer

Name	Type	Size	Value
mi_lista	list	2	[4, 8]

IPython console

Console 1/A X Console 2/A X

Python 3.6.9 [Anaconda, Inc.] (default, Jul 30 2019, 19:07:31)
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

```

In [1]: mi_lista = []

In [2]: mi_lista.append(4)

In [3]: mi_lista.append(8)

In [4]: mi_lista.append(12)

In [5]: mi_lista
Out[5]: [4, 8, 12]

In [6]: mi_lista.pop()
Out[6]: 12

```

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
- Así como `append`, hay **muchas** funciones que se pueden utilizar.

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
- Así como `append`, hay **muchas** funciones que se pueden utilizar.
- Permiten definir cierto **comportamiento interesante** y no tener que volverlo a escribir cada vez.

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
- Así como `append`, hay **muchas** funciones que se pueden utilizar.
- Permiten definir cierto **comportamiento interesante** y no tener que volverlo a escribir cada vez.
- Todos los lenguajes de programación **tienen** un mecanismo para definir funciones.

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
- Así como `append`, hay **muchas** funciones que se pueden utilizar.
- Permiten definir cierto **comportamiento interesante** y no tener que volverlo a escribir cada vez.
- Todos los lenguajes de programación **tienen** un mecanismo para definir funciones.
- Los valores que recibe una función se llaman **parámetros** o **argumentos**:

Funciones

- Una construcción que permite *encerrar* un *pedacito* de programa. En un libro de recetas se definen muchas funciones: se cuenta la primera vez cómo se hace algo (ejemplo: cómo batir dos claras a punto nieve) y en el resto de las recetas simplemente se dice “bata dos claras a punto nieve”).
- Así como `append`, hay **muchas** funciones que se pueden utilizar.
- Permiten definir cierto **comportamiento interesante** y no tener que volverlo a escribir cada vez.
- Todos los lenguajes de programación **tienen** un mecanismo para definir funciones.
- Los valores que recibe una función se llaman **parámetros** o **argumentos**:

Ejemplo: Discriminante de una cuadrática $p(x) = ax^2 + bx + c$:

$$\text{Raíces: } \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$\Delta(a, b, c) = b^2 - 4ac > 0$: 2 raíces reales

$\Delta(a, b, c) = b^2 - 4ac = 0$: raíz real doble

$\Delta(a, b, c) = b^2 - 4ac < 0$: raíces complejas conjugadas

Funciones: a probarlo

```
def discriminante(a, b, c):  
    x = b * b  
    y = 4 * a * c  
    r = x - y  
    return r  
res = discriminante(1, 0, -1)  
print(res)  
r = discriminante(1, 0, 1)  
print(r)
```

Tabulación

Python *sabe* donde termina la definición de una función por la **tabulación**. En el caso de la función `discriminante` las instrucciones que componen la función están cuatro espacios hacia la izquierda.

Strings: listas raras de caracteres

- Vimos algunos ejemplos de listas con números. Veamos ahora que podemos definir también estructuras con caracteres.
- Para definir un *string* (cadena de caracteres), podemos hacer esto:

```
nombre = "Marianito"
```

Esto define la variable `nombre` que contiene una cadena de caracteres.

Strings: listas raras de caracteres

- Vimos algunos ejemplos de listas con números. Veamos ahora que podemos definir también estructuras con caracteres.
- Para definir un *string* (cadena de caracteres), podemos hacer esto:

```
nombre = "Marianito"
```

Esto define la variable `nombre` que contiene una cadena de caracteres.

- Podemos imprimir el string:

```
print(nombre)
```

Strings: listas raras de caracteres

- Vimos algunos ejemplos de listas con números. Veamos ahora que podemos definir también estructuras con caracteres.
- Para definir un *string* (cadena de caracteres), podemos hacer esto:

```
nombre = "Marianito"
```

Esto define la variable `nombre` que contiene una cadena de caracteres.

- Podemos imprimir el string:

```
print(nombre)
```

- Pero podemos hacerlo mucho más divertido:

```
print("Mi nombre es:" + nombre)
```

El resultado de esto es que, en pantalla, vamos a ver el contenido de dos cadenas... ¿Cuáles? ¡Probémoslo!

Strings: listas raras de caracteres

- Vimos algunos ejemplos de listas con números. Veamos ahora que podemos definir también estructuras con caracteres.
- Para definir un *string* (cadena de caracteres), podemos hacer esto:

```
nombre = "Marianito"
```

Esto define la variable `nombre` que contiene una cadena de caracteres.

- Podemos imprimir el string:

```
print(nombre)
```

- Pero podemos hacerlo mucho más divertido:

```
print("Mi nombre es:" + nombre)
```

El resultado de esto es que, en pantalla, vamos a ver el contenido de dos cadenas... ¿Cuáles? ¡Probémoslo!

- Una vez que lo probemos y si no cambiamos nada, lo que salió en pantalla fue:

```
Mi nombre es:Marianito
```

Se ve medio feo, ¿no? Está pegoteado el nombre y ":". ¿Cómo lo arreglamos?

Metiéndose con las cadenas

Hay algunas operaciones interesantes para hacer con cadenas (también con listas en general):

- `len()`: es una función que recibe una cadena de caracteres (o una lista) y te devuelve la cantidad de elementos que tiene (viene de “length”, longitud).
Por ejemplo la podemos usar `len(lista_de_las_compras)` y nos dirá el largo de la lista que le pasamos.
`len([])` da...

Metiéndose con las cadenas

Hay algunas operaciones interesantes para hacer con cadenas (también con listas en general):

- `len()`: es una función que recibe una cadena de caracteres (o una lista) y te devuelve la cantidad de elementos que tiene (viene de “length”, longitud). Por ejemplo la podemos usar `len(lista_de_las_compras)` y nos dirá el largo de la lista que le pasamos.
`len([])` da...¡0!

Metiéndose con las cadenas

Hay algunas operaciones interesantes para hacer con cadenas (también con listas en general):

- `len()`: es una función que recibe una cadena de caracteres (o una lista) y te devuelve la cantidad de elementos que tiene (viene de “length”, longitud). Por ejemplo la podemos usar `len(lista_de_las_compras)` y nos dirá el largo de la lista que le pasamos.

`len([])` da...¡0!

- `[]`: es un *operador* para acceder a cualquier elemento de una cadena o de una lista:

```
nombre = "Marianito"
print(nombre[0])
print(nombre[5])
```

Hay que tener cuidado con acceder a los elementos fuera de rango, aparecen errores feos.

Metiéndose con las cadenas

Más cosas:

- También se pueden crear listas usando `[]` y dando los elementos:

```
lista_numeros = [1,2,3,10,99]
```

```
lista_caracteres = ['1','2','3','10','99']
```

Metiéndose con las cadenas

Más cosas:

- También se pueden crear listas usando `[]` y dando los elementos:

```
lista_numeros = [1,2,3,10,99]
```

```
lista_caracteres = ['1','2','3','10','99']
```

- `+`: concatenación, toma dos strings (o listas) y los *pega*:

```
apellido = "Quito"
```

```
primer_nombre = "Armando"
```

```
segundo_nombre = "Esteban"
```

```
print("Hola!!, quiero hablar con " + apellido + ", " + primer_nombre  
      + segundo_nombre + apellido + " por favor")
```

¿Funciona bien? ¿Se puede mejorar?

Los strings no son listas, son parecidos

- Si bien los strings son *casi* como listas, son **inmutables**.
- Esto significa que, una vez definidos, no se pueden cambiar:

```
nombre = "Marianito"  
nombre[8] = 'a'
```

Esto **no** funciona da un error que dice

`TypeError: 'str' object does not support item assignment.`
Significa que estamos queriendo cambiar algo que **no** se puede cambiar.

Los strings no son listas, son parecidos

- Si bien los strings son *casi* como listas, son **inmutables**.
- Esto significa que, una vez definidos, no se pueden cambiar:

```
nombre = "Marianito"  
nombre[8] = 'a'
```

Esto **no** funciona da un error que dice

`TypeError: 'str' object does not support item assignment.`
Significa que estamos queriendo cambiar algo que **no** se puede cambiar.

- La manera para trabajar esto es usando una lista de verdad:

```
nombre = "Marianito"  
milista = list(nombre)  
milista[8] = 'a'
```

Los strings no son listas, son parecidos

- Si bien los strings son *casí* como listas, son **inmutables**.
- Esto significa que, una vez definidos, no se pueden cambiar:

```
nombre = "Marianito"  
nombre[8] = 'a'
```

Esto **no** funciona da un error que dice

`TypeError: 'str' object does not support item assignment.`
Significa que estamos queriendo cambiar algo que **no** se puede cambiar.

- La manera para trabajar esto es usando una lista de verdad:

```
nombre = "Marianito"  
milista = list(nombre)  
milista[8] = 'a'  
print(milista)  
print(nombre)
```


Los strings no son listas, son parecidos

- Si bien los strings son *casí* como listas, son **inmutables**.
- Esto significa que, una vez definidos, no se pueden cambiar:

```
nombre = "Marianito"  
nombre[8] = 'a'
```

Esto **no** funciona da un error que dice

`TypeError: 'str' object does not support item assignment.`
Significa que estamos queriendo cambiar algo que **no** se puede cambiar.

- La manera para trabajar esto es usando una lista de verdad:

```
nombre = "Marianito"  
milista = list(nombre)  
milista[8] = 'a'  
print(milista)  
print(nombre)
```

```
milista[len(milista)-1] = 'a' # Alternativa para lo mismo  
#(esto es un comentario)
```

Ciclos

Quiero imprimir en pantalla los números del 1 al 10:

Ciclos

Quiero imprimir en pantalla los números del 1 al 10:

```
print(1)  
print(2)  
print(3)  
print(4)  
print(5)
```

```
print(6)  
print(7)  
print(8)  
print(9)  
print(10)
```

¡Qué gran programa!

Ciclos

Quiero imprimir en pantalla los números del 1 al 10:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
print(6)
print(7)
print(8)
print(9)
print(10)
```

¡Qué gran programa!

Bueno, bueno, dejen de tirar cosas y abuchear, lo podemos hacer así:

```
i = 1
while i <= 10:
    print(i)
    i = i+1
```

Ciclos

Quiero imprimir en pantalla los números del 1 al 10:

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

```
print(6)
print(7)
print(8)
print(9)
print(10)
```

¡Qué gran programa!

Bueno, bueno, dejen de tirar cosas y abuchear, lo podemos hacer así:

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

Si quisiéramos sumar 1 a todos los elementos de una lista:

```
def cambiar_lista(lis):
    i = 0
    while i < len(lis):
        lis[i] = lis[i] + 1
        i = i + 1
    return lis
```

Escribamos los dos programas en el tutor y veamos cómo funcionan. En el segundo caso, no se olviden de **llamar** a la función con algún parámetro **razonable** para probarla.

Otra estructura de control: `if`

- Permite ejecutar una serie de instrucciones si se cumple cierta condición.

Otra estructura de control: `if`

- Permite ejecutar una serie de instrucciones si se cumple cierta condición.
- Supongamos que queremos cambiar un valor dependiendo si vale o no 2:

```
mivalor = 2
if mivalor==2:
    mivalor = 5
else:
    mivalor = 8

print(mivalor)
```

Otra estructura de control: `if`

- Permite ejecutar una serie de instrucciones si se cumple cierta condición.
- Supongamos que queremos cambiar un valor dependiendo si vale o no 2:

```
mivalor = 2
if mivalor==2:
    mivalor = 5
else:
    mivalor = 8

print (mivalor)
```

Los dos puntos (:) son obligatorios,

¡No olvidarse!

Veamos un ejemplo más divertido:

```
def CuantasRaices(a,b,c):  
    d=discriminante(a,b,c)  
    if d > 0:  
        res = 2  
        print("El polinomio tiene dos raices reales diferentes.")  
    elif d == 0:  
        res = 1  
        print("El polinomio tiene una raiz real doble.")  
    else:  
        res = 0  
        print("El polinomio no tiene raices reales.")  
    return res  
  
prueba=CuantasRaices(1,0,-1)  
print(prueba)
```

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - < menor

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - < menor
 - <= menor o igual
 - > mayor
 - >= mayor o igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - < menor
 - <= menor o igual
 - > mayor
 - >= mayor o igual
 - == igual

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - < menor
 - <= menor o igual
 - > mayor
 - >= mayor o igual
 - == igual
 - != distinto

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - < menor
 - <= menor o igual
 - > mayor
 - >= mayor o igual
 - == igual
 - != distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.

Comparaciones y condiciones

- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.

Comparaciones y condiciones

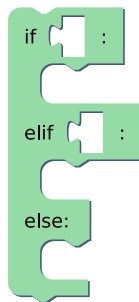
- Se pueden realizar distintas comparaciones:
 - `<` menor
 - `<=` menor o igual
 - `>` mayor
 - `>=` mayor o igual
 - `==` igual
 - `!=` distinto
- También se pueden combinar distintas condiciones utilizando los operadores lógicos:
 - **not** negación, si se aplica a `True`, da `False` y a la inversa.
 - **and** se usa `x and y`. Solo da `True` cuando `x` e `y` son `True`.
 - **or** se usa `x or y`. Da `True` cuando alguna de las dos (o las dos) es `True`.
- Esto aplica tanto para las condiciones del `if` como a las del `while`.

```
if a>x and a<y:  
    c = x*x+y*y  
else:  
    c = 2*x*y
```

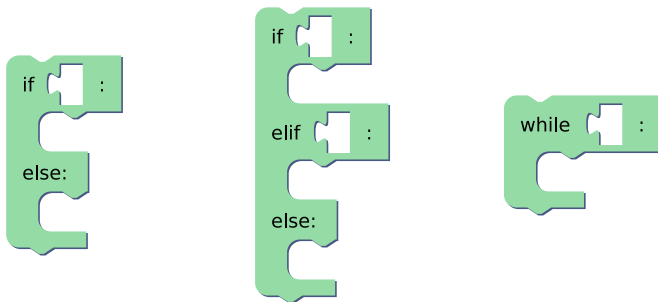
Estructuras



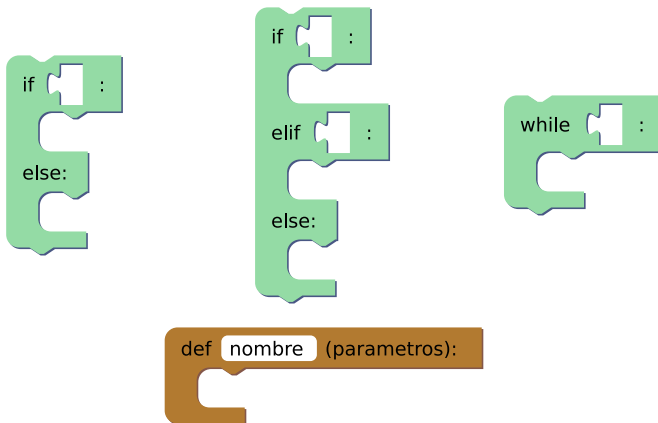
Estructuras



Estructuras



Estructuras



Ejercicios

Ejercicio

Defina una función que recibe una cadena de caracteres y devuelve la cantidad de letras e que contiene.

```
def cant_e ----:
    i=0
    count=0
    while i<----
        if ----[i]=='e'----
            count=count+1
        i+=1
    return ----
```

Sutilezas surtidas

Conversión entre tipos de datos

```
int(3.5)  
int(2.999)  
int(-2.3)  
list('palabra')
```

Sutilezas surtidas

Conversión entre tipos de datos

```
int(3.5)
int(2.999)
int(-2.3)
list('palabra')
```

Entrada de datos

```
nombre=input("Nombre: ")
edad=int(input("Edad: "))
```

Sutilezas surtidas

Conversión entre tipos de datos

```
int(3.5)
int(2.999)
int(-2.3)
list('palabra')
```

Entrada de datos

```
nombre=input("Nombre: ")
edad=int(input("Edad: "))
```

Los Keywords no pueden usarse como nombres de variables

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	

Sutilezas surtidas

Conversión entre tipos de datos

```
int(3.5)
int(2.999)
int(-2.3)
list('palabra')
```

Entrada de datos

```
nombre=input("Nombre: ")
edad=int(input("Edad: "))
```

Los Keywords no pueden usarse como nombres de variables

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	

Comentarios útiles e inútiles

```
v = 5    #le asigno 5 a v
v = 5    #velocidad en metros/seg
```

Sutilezas surtidas

Conversión entre tipos de datos

```
int(3.5)
int(2.999)
int(-2.3)
list('palabra')
```

Entrada de datos

```
nombre=input("Nombre: ")
edad=int(input("Edad: "))
```

Los Keywords no pueden usarse como nombres de variables

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	

Comentarios útiles e inútiles

```
v = 5    #le asigno 5 a v
v = 5    #velocidad en metros/seg
```

Diferencias entre modo interactivo y modo script

Sutilezas surtidas

Las variables locales (y los parámetros) son locales

```

1 # -*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8
9
10 def concaternar_e_imprimir(p1,p2):
11     conc=p1+p2
12     print(conc)
13     print(conc)
14
15 concaternar_e_imprimir("abra","cadabra")
16
17 print(conc)
18

```

Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 19:07:31)
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: def concaternar_e_imprimir(p1,p2):
...: conc=p1+p2
...: print(conc)
...: print(conc)
...:
...:

In [2]: concaternar_e_imprimir("abra","cadabra")
abracadabra
abracadabra

In [3]: print(conc)
Traceback (most recent call last):

File "<ipython-input-3-296c77c0a816>", line 1, in <module>
 print(conc)
NameError: name 'conc' is not defined

Sutilezas surtidas

Las variables locales (y los parámetros) son locales

The image shows a Spyder IDE window with a script editor on the left and an IPython console on the right.

Script Editor (Left):

```

1 #-*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8
9
10 def concatenar_e_imprimir(p1,p2):
11     conc=p1+p2
12     print(conc)
13     print(conc)
14
15 concatenar_e_imprimir("abra","cadabra")
16
17 print(conc)
18

```

IPython Console (Right):

```

Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 19:07:31)
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: def concatenar_e_imprimir(p1,p2):
...:     conc=p1+p2
...:     print(conc)
...:     print(conc)
...:
...:

In [2]: concatenar_e_imprimir("abra","cadabra")
abracadabra
abracadabra

In [3]: print(conc)
Traceback (most recent call last):

  File "<ipython-input-3-296c77c0a816>", line 1, in <module>
    print(conc)
NameError: name 'conc' is not defined

```

Composicion de funciones y el flujo de la ejecucion

```

palabra = "Pyhton"
concatenar_e_imprimir(palabra.upper(), palabra.lower())

```

Sutilezas surtidas

Las variables locales (y los parámetros) son locales

The image shows a Spyder IDE window with a script editor on the left and an IPython console on the right.

Script Editor (Left):

```

1 #-*- coding: utf-8 -*-
2 """
3 Spyder Editor
4
5 This is a temporary script file.
6 """
7
8
9
10 def concatenar_e_imprimir(p1,p2):
11     conc=p1+p2
12     print(conc)
13     print(conc)
14
15 concatenar_e_imprimir("abra","cadabra")
16
17 print(conc)
18

```

IPython Console (Right):

```

Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 19:07:31)
Type "copyright", "credits" or "license" for more information.

IPython 6.2.1 -- An enhanced Interactive Python.

In [1]: def concatenar_e_imprimir(p1,p2):
...:     conc=p1+p2
...:     print(conc)
...:     print(conc)
...:
...:

In [2]: concatenar_e_imprimir("abra","cadabra")
abracadabra
abracadabra

In [3]: print(conc)
Traceback (most recent call last):

  File "<ipython-input-3-296c77c0a816>", line 1, in <module>
    print(conc)
NameError: name 'conc' is not defined

```

Composicion de funciones y el flujo de la ejecucion

```

palabra = "Pyhton"
concatenar_e_imprimir(palabra.upper(),palabra.lower())

```

Links útiles

El Google Drive de Programación en Python es:

`http://bit.ly/UnsamPython2020C1`

Allí van a encontrar

- El link de invitación al grupo de WhatsApp de la materia.
- Esta clase y la correspondiente guía de ejercicios.
- El formulario de autoevaluación de la guía 1.
- En un futuro, más material del curso....