Universidad Tecnológica de Pereira

Misión TIC 2022

El futuro digital es de todos

MinTIC

# Depuración y Módulo

Hechos QUE CONECTAN

# Contenido

Depuración y Módulos

https://imaster.academy/course/view.php?id=367&section=1

# Lógica e Interacción

# Contenido

- Introducción a la programación organizada: Módulos
- Lógica vs. Interacción
- Separación de la lógica y la interfaz
- Importando módulos
- Nuestro primer módulo
- Manipulación de entrada y salida de información en un programa: interfaces por consola
- Depuración de código en VSC

# Contenido

## Introducción a la programación organizada: Módulos

- La programación modular es una técnica utilizada para separar las funcionalidades de un programa, organizándolas en módulos independientes. Esto es bastante útil cuándo se trabaja en proyectos a gran escala ya que nos permite seccionar el código de manera que sea más fácil de utilizar y leer. Por ejemplo, es mucho mejor que tener mil declaraciones de funciones en un solo archivo!
- Adicionalmente, Python es un lenguaje OpenSource en el que sus usuarios de todo el mundo son libres de compartir sus propias funciones y módulos. Esto nos permitirá acceder y utilizar una infinidad de librerías públicas, ahorrandonos tiempo, esfuerzo y dinero.

# Contenido

## Lógica vs. Interacción

- La mayor preocupación para quien desarrolla un sistema debería ser la mantenibilidad. Esto quiere decir: qué tan fácil es hacerle un cambio a un sistema o programa (ej. agregarle alguna funcionalidad, corregir algún error, mejorar su apariencia).
- Cuando se está aprendiendo a programar puede parecer que esta no es una cuestión importante y que lo prioritario son la corrección y el desempeño. La realidad es que una vez se empieza a usar un programa, siempre es necesario hacerle modificaciones, especialmente cuando es muy exitoso. Sólo piense en la cantidad de actualizaciones que tiene que instalar permanentemente en su computador o en su celular: con casi total seguridad usted no está utilizando la primera versión (1.0) de ninguna aplicación.

# Contenido

## Lógica vs. Interacción

La pregunta natural que debería surgir es entonces: ¿cómo logramos que un programa sea mantenible?

- Documentar el código. Cuando el código está documentado es más fácil que otros programadores o nosotros mismos podamos hacer mejoras o correcciones a nuestros programas.
- Estructurar el código. Estructuras el código significa hacer el programa fácil de entender. Esto significa que queremos que sea fácil entender cómo está organizado para que cuando haya un error sea fácil encontrar donde se debería corregir, o para que cuando se quiera agregar nuevas funcionalidades no se necesite una gran reestructuración.

# Contenido

Separación de la lógica y la interfaz

??

**Dividir para organizar acciones diferentes**

**Interfaces, cálculos, mensajes, JUNTOS????**

**Coherencia, Mantenimiento, Reutilización**

# Contenido

## Importando un módulo

Para que un módulo sea utilizable, hay que importarlo (piensa en ello como sacar un libro del estante). La importación de un módulo se realiza mediante una instrucción llamada `import`. Nota: `import` es también una palabra reservada (con todas sus implicaciones).
La forma más sencilla de importar un módulo en particular es usar la instrucción de importación de la siguiente manera:

```
import math
```

La cláusula contiene:

- La palabra reservada import.
- El nombre del módulo que se va a importar.

# Contenido

## Importante…

La instrucción puede colocarse en cualquier parte del código, pero debe colocarse antes del primer uso de cualquiera de las entidades del módulo.

Si se desea (o se tiene que) importar más de un módulo, se puede hacer repitiendo la cláusula `import`, o listando los módulos después de la palabra reservada `import`, por ejemplo:

```
import math, sys
```

La instrucción importa dos módulos, primero uno llamado `math` y luego un segundo llamado `sys`.

# Contenido

## namespace

Un **namespace** es un espacio (entendido en un contexto no físico) en el que existen algunos nombres y los nombres no entran en conflicto entre sí (es decir, no hay dos objetos diferentes con el mismo nombre). Podemos decir que cada grupo social es un namespace - el grupo tiende a nombrar a cada uno de sus miembros de una manera única (por ejemplo, los padres no darán a sus hijos los mismos nombres).

**Dentro de un determinado namespace, cada nombre debe permanecer único**. Esto puede significar que algunos nombres pueden desaparecer cuando cualquier otra entidad de un nombre ya conocido ingresa al namespace.

Si el módulo de un nombre especificado **existe y es accesible** (un módulo es de hecho un **archivo fuente de Python**), Python importa su contenido, **se hacen conocidos todos los nombres definidos en el módulo**, pero no ingresan al namespace del código.

# Contenido

## Cómo acceder al pi el cual viene del módulo math.

Para hacer esto, se debe de mandar llamar el pi con el su nombre en el módulo original.

Observa el fragmento a continuación, esta es la forma en que se habilitan los nombres de `pi` y `sin` con el nombre de su módulo de origen:

```
import math
math.pi
math.sin
```

# Contenido

## Cómo acceder al pi el cual viene del módulo math.

En el **segundo método**, la sintaxis del import señala con precisión qué entidad (o entidades) del módulo son aceptables en el código:

```
from math import pi
```

**La instrucción consta de los siguientes elementos:**

- La palabra reservada from.
- El **nombre del módulo** a ser (selectivamente) importado.
- La palabra reservada import.

El **nombre o lista de nombres de la entidad o entidades** las cuales estan siendo importadas al namespace.

# Contenido

## Cómo acceder al pi el cual viene del módulo math.

En el tercer método, la sintaxis del import es una forma más agresiva que la presentada anteriormente:

```
from module import *
```

Como puedes ver, el nombre de una entidad (o la lista de nombres de entidades) se reemplaza con un solo asterisco (*).
Tal instrucción importa todas las entidades del módulo indicado.

# Contenido

## la palabra reservada as

Si se importa un módulo y no se esta conforme con el nombre del módulo en particular (por ejemplo, sí es el mismo que el de una de sus entidades ya definidas) puede dársele otro nombre: esto se llama aliasing o renombrado.
Aliasing (renombrado) hace que el módulo se identifique con un nombre diferente al original.
La creación de un alias se realiza junto con la importación del módulo, y exige la siguiente forma de la instrucción import:

```
import module as alias
```

El "module" identifica el nombre del módulo original mientras que el "alias" es el nombre que se desea usar en lugar del original.
Nota: as es una palabra reservada.

# Contenido

## la palabra reservada as

A su vez, cuando usa la variante `from module import name` y se necesita cambiar el nombre de la entidad, se crea un alias para la entidad. Esto hará que el nombre sea reemplazado por el alias que se elija.

Así es como se puede hacer:

```
from module import nombre as alias
from module import n as a, m as b, o as c
from math import pi as PI, sin as sine
```

# Contenido

## la palabra reservada as

A su vez, cuando usa la variante `from module import name` y se necesita cambiar el nombre de la entidad, se crea un alias para la entidad. Esto hará que el nombre sea reemplazado por el alias que se elija.
Así es como se puede hacer:

```
#Main
from module import nombre as alias
from module import n as a, m as b, o as c

alias()
print(a(), b(), c())
print(a() + b() + c())


from math import pi as PI, sin as
sine print(sine(PI/2))
```

```
#Module
def nombre():
   miNombre = "Luis"
   print(miNombre)

def n():
   return 2

def m():
   return 10

def o():
   return 2
```

# Contenido

## Ejemplo!!

```
from math import pi as PI, sin as sine

print(sine(PI/2))

import math as m

print(m.sin(m.pi/2))

from random import random
print(random())
```

```
from random import *

print(randrange(1), end = ' ')

print(randrange(0, 20), end=' ')

print(randrange(0, 5, 1), end=' ')

print(randint(0, 1))
```

**randrange(fin)**

**randrange(inico, fin)**

**randrange(inicio, fin, incremento)**

**randint(izquierda, derecha)**

# Contenido

Ejemplo!!

```
from platform import*

print(version())
print(platform())
print(platform(1))
print(platform(0, 1))
print(processor())
print(system())
```

**módulos estándar de Python**

**aquí: https://docs.python.org/3/py-modindex.html.**

# Contenido

## Nuestro primer modulo

```
#modulo Main.py
import Operacion
print(Operacion.suma(4, 5))

#modulo Operacion
def suma (x, y):
      return x+y
```

# Contenido

## Ejemplo!!!

```
#modulo Main.py
import Mensajes

Mensajes.mensajeBienvenida()

#modulo Mensajes.py

def mensajeBienvenida():
    print("Bienvenido a la introducción a módulos")
```

# Contenido

## Ejemplo!!!

```
#modulo Main.py
import Logica

nombre = input("¿Ingrese su nombre? ")

texto_saludo = Logica.saludo(nombre)

print(texto_saludo)
```

```
#modulo Logica.py

def saludo(cadena):
    return "Hola {}! ¿cómo estas?".format(cadena)
```

# Contenido

## Ejemplo!!!

```
#modulo Main.py
import Logica as Saludos

nombre = input("¿Ingrese su nombre? ")

texto_saludo = Saludos.saludo(nombre)

print(texto_saludo)
```

```
#modulo Logica.py

def saludo(cadena):
    return "Hola {}! ¿cómo estas?".format(cadena)
```

# Contenido

## Ejemplo!!!

```
#modulo Main
import Operacion

numero1 = int(input("Ingrese el numero 1 "))
numero2 = int(input("Ingrese el numero 2 "))

Operacion.resultado(numero1, numero2)
```

```
#modulo Operacion
def resultado(numero1, numero2):
    print("La suma es igual a :", numero1 +
numero2)
    print("La resta es igual a :", numero1 -
numero2)
```

# Contenido

## Manipulación de entrada y salida de información en un programa: interfaces por consola

### Función print(" ")

Una función (en este contexto) es una parte separada del código de computadora el cual es capaz de:

- Causar algún efecto (por ejemplo, enviar texto a la terminal, crear un archivo, dibujar una imagen, reproducir un sonido, etc.); esto es algo completamente inaudito en el mundo de las matemáticas.
- Evaluar un valor o algunos valores (por ejemplo, la raíz cuadrada de un valor o la longitud de un texto dado); esto es lo que hace que las funciones de Python sean parientes de los conceptos matemáticos.

¿De dónde provienen las funciones?: de python o de sus complementos (módulos), o de complementos externos que deben estar embebidos en el código

Como se dijo anteriormente, una función puede tener:

- Un efecto.
- Un resultado.

Python requiere que no haya más de una instrucción por una línea.

# Contenido

## Función print("  ") - Parametros

\n agrega una línea en blanco, donde "\" es el carácter de escape
print("\\") inserta la "\"

**Importante**

- los argumentos se separan con "coma" como el c#
- Un argumento de palabra clave consta de tres elementos: una palabra clave que identifica el argumento (end -termina aquí); un signo de igual (=); y un valor asignado a ese argumento.
- Cualquier argumento de palabra clave debe ponerse después del último argumento posicional (esto es muy importante).

# Contenido

## Ejemplos!!!

```
print("Mi nombre es", "Python.", end="    ")
print("Mi nombre es", "Python.", end="\n")
print("Mi nombre es ", end="")
print("Mi", "nombre", "es", "Monty", "Python.", sep="-")
print("Mi", "nombre", "es", "Monty", "Python.", sep="    ")
```

# Contenido

## Ejemplos!!!

```
print("Mi", "nombre", "es", sep="_", end="*")
print("Monty", "Python.", sep="*", end="*\n")
print("Fundamentos","Programación","en", sep="_", end="_")
```

# Contenido

## Ejemplos!!!

Conversión de octal a decimal

```
print(0o123)
```

Conversión de octal a decimal

```
print(0x123)
```

# Contenido

## Ejemplos!!!

```
print("Me gusta \"Monty Python\"")
print('Me gusta "Monty Python')
print(True > False)
print(True < False)
print('\"Estoy\" \n\"\"aprendiendo\"\" \n\"\"\"Python\"\"\"')
```

# Contenido

## Ejemplos!!!

```
print(2 ** 3)
print(2 ** 3.)
print(2. ** 3)
print(2. ** 3.)
print(2 * 3.)
print(2. * 3)
print(2. * 3.)
print(6 / 3)
print(6 / 3.)
print(6. / 3)
print(6. / 3.)
print(6 // 3)
```

```
print(6 // 3.)
print(6. // 3)
print(6. // 3.)
print(6 // 4)
print(6. // 4)
print(-6 // 4)
print(6. // -4)
print(14%4.5)
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
print((2**4), (2*4.), (2*4))
print((-2/4), (2/4), (2//4), (-2//4))
print((2%-4), (2%4), (2**3**2))
```

# Contenido

## Ejemplos!!!

```
nom = "Luis"
ape = "Molero"
print("\n Tu nombre es " + nom + " " + ape + ".")

print("James" * 3)
print(5 * "2")
print("2" * 5)

nmasGrande = 9.56874
print("El número más grande es:", round(nmasGrande))
```

# Contenido

## La función input()

```
print("x > 5")
print("x == 10")
print("\nLa instrucción continua:")

entrada = 'Hola'
print ("La palabra ingresada fue", entrada)
```

# Contenido

## La función input()

```
a, b, c, d, e, f  = 10, 15, 25, 68, 999, 556
print(a, b, c, d, e, f)

numeros = [10, 5, 7, 2, 1]
print("Contenido de la lista original:", numeros)

numeros = [10, 5, 7, 2, 1]
numeros[0] = 111
print("\nPrevio contenido de la lista:", numeros)
```

# Contenido

## La función input()

```
numeros = [10, 5, 7, 2, 1]
print(numeros)
numeros[1] = numeros[4]
print("Nuevo contenido de la lista:", numeros)
```

# Contenido

La función input()

Tipos de datos:

int() y float().

# Contenido

## La función input()

```
algo = float(input("Inserta un número: "))
resultado = algo ** 2.0
print(algo, "al cuadrado es", resultado)

nom = input("¿Me puedes dar tu nombre por favor? ")
ape = input("¿Me puedes dar tu apellido por favor? ")
print("Gracias.")
```

# Contenido

## La función input()

```
algo = int(input("Inserta un número entero: "))
resultado = algo * 2
print(algo, " x 2 es ", resultado)

nom = input("¿Me puedes dar tu nombre por favor? ")
ape = input("¿Me puedes dar tu apellido por favor? ")
print("Gracias.")
```