Fundamentos de programación en PYTHON

Carrera de Administración de Redes y Comunicaciones

InGenio Learning

Docente: Mg. Gustavo Coronel

Alianzas Estratégicas

Microsoft Azure

aws educate

Google Meet

Cisco Networking Academy

Google Classroom

Kahoot!

nova cloud

ORACLE ACADEMY

FORTINET NSE Training Institute
Authorized Security Academy

RED HAT ACADEMY

CERTIPORT
A PEARSON VUE BUSINESS
AUTHORIZED TESTING CENTER

Pearson VUE
Authorized Test Center

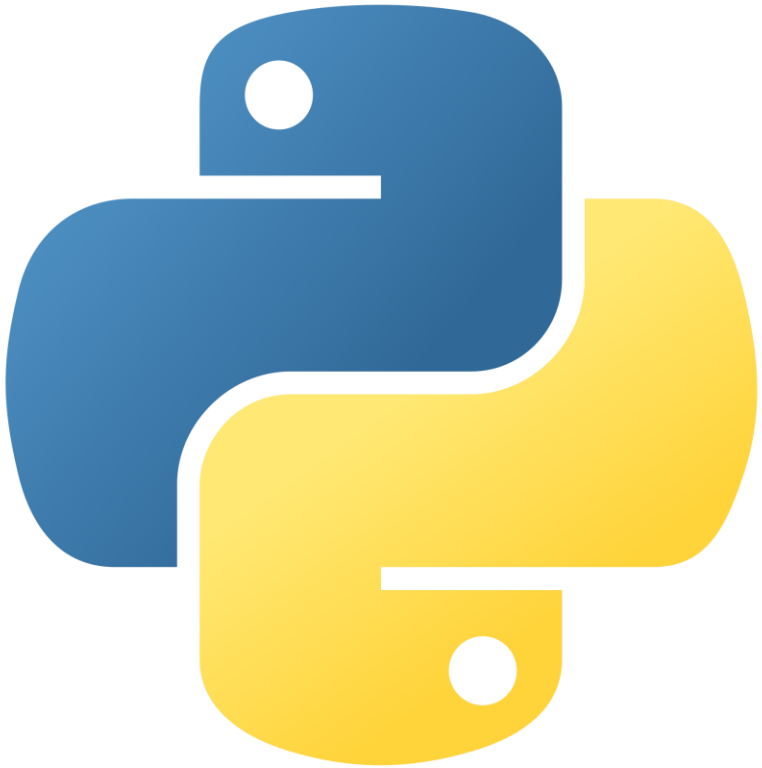
win E

Google Workspace

Office 365

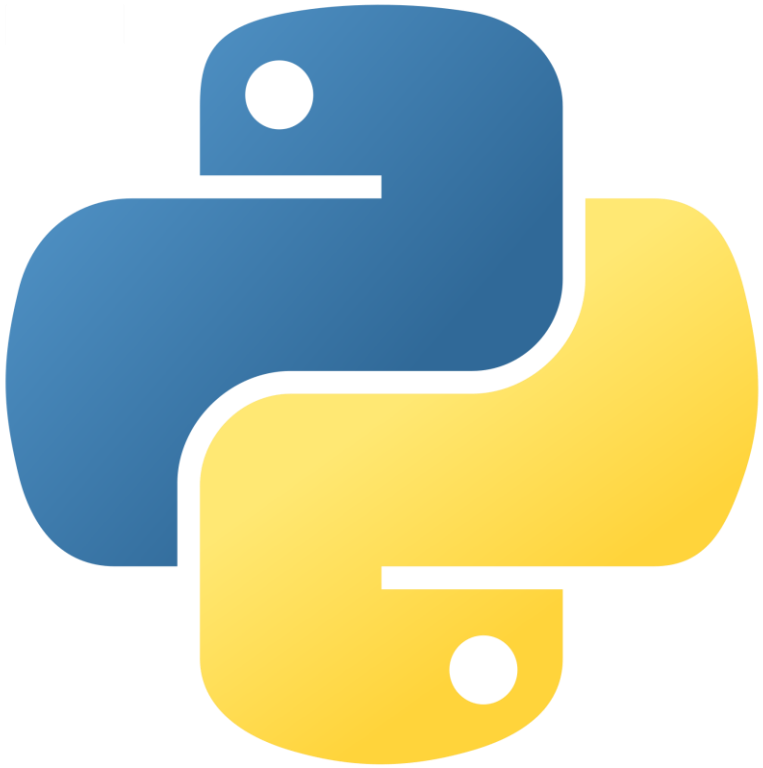
Q10
Académico

Sobre el curso



El curso está orientado a la enseñanza de fundamentos de Programación en el lenguaje Python v 3.0., bajo la modalidad definida por la Academia de Programming de Cisco. Esta modalidad consiste en el desarrollo de la currícula PCAP dividida en 8 módulos.

Durante el curso se desarrollarán actividades de revisión del contenido académico y laboratorios prácticos utilizando el emulador de Python que nos provee el mismo Cisco (SandBox) o de algún otro IDE para escritorio, así como el desarrollo de exámenes y actividades orientadas al entendimiento de estos conceptos de acuerdo con los temas que considera Cisco en este curso.



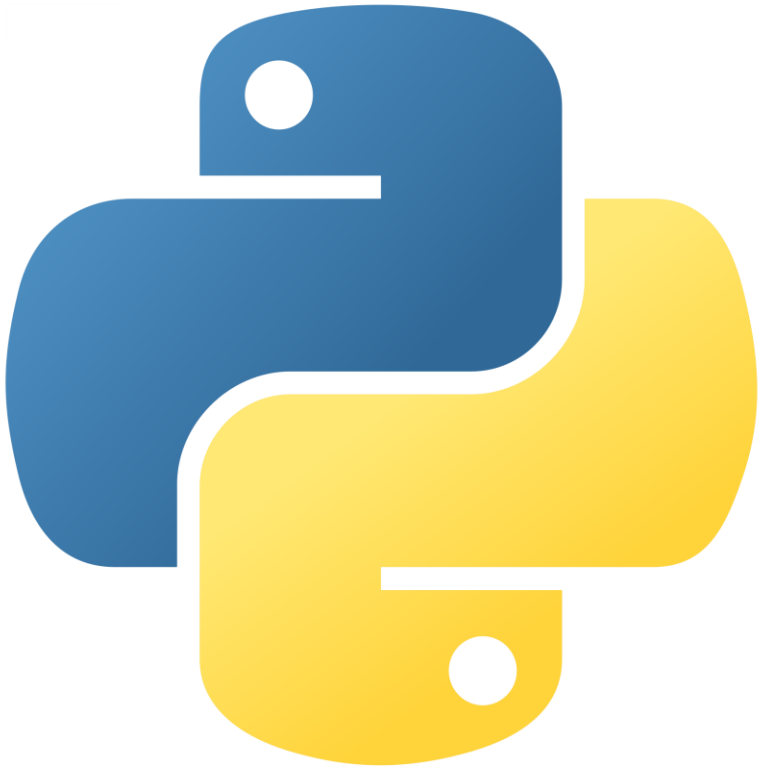
SEMANA 7

MODULOS, PAQUETES Y EXCEPCIONES

LOGRO DE LA SESION

Al finalizar la sesión, aprenderás a desarrollar y utilizar módulos, .

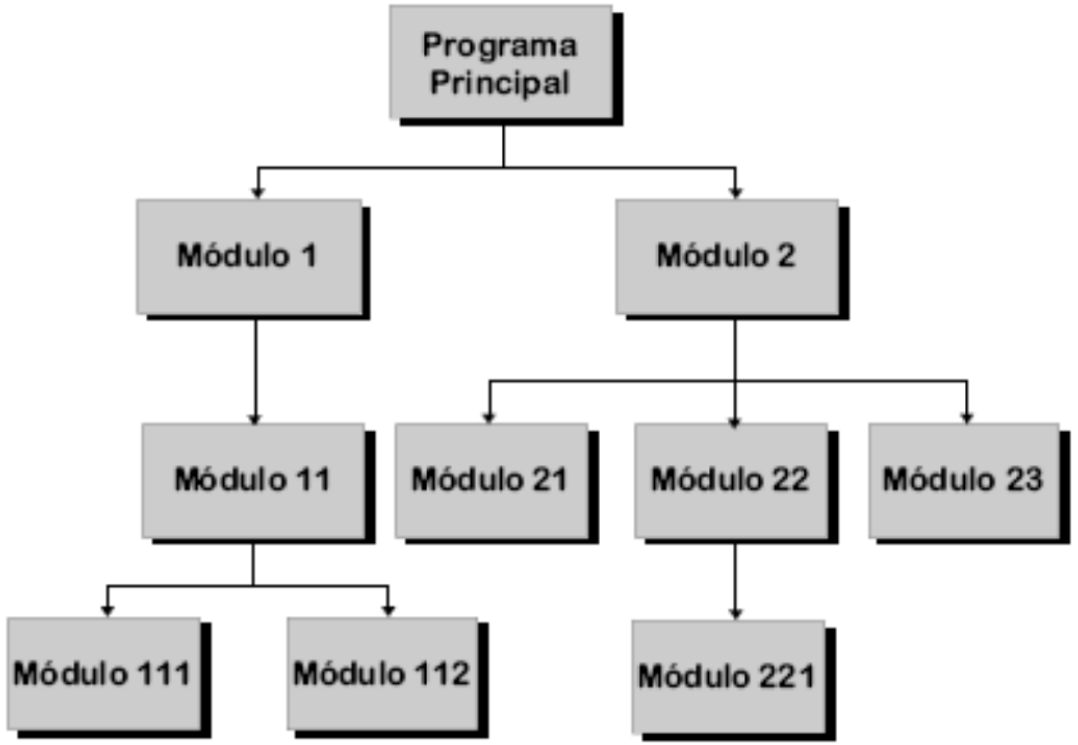
Contenido



- Introducción
- Módulos
- Paquetes
- Excepciones
- Conclusiones
- Evaluación Continua 3

Introducción

La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.



Módulo

Es un archivo de código fuente que contiene definiciones de funciones, clases, variables y otros objetos que pueden ser utilizados en un programa.

Los módulos permiten a los programadores organizar su código en unidades lógicas y reutilizar código en diferentes programas.

```
import math
resultado = math.sqrt(16)
print(resultado)
```

Módulo

Importando el módulo

```
import modulo1
print(modulo1.suma(8,6))
print(modulo1.resta(8,6))
print(modulo1.multiplica(8,6))
```

modulo1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Módulo

Renombrando el módulo

```
import modulo1 as mo
print(mo.suma(8,6))
print(mo.resta(8,6))
print(mo.multiplica(8,6))
```

modulo1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Módulo

Importando solo una función

```
from modulo1 import suma
print(suma(8,6))
```

modulo1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Módulo

Importando todo el contenido

```
from modulo1 import *
print(suma(8,6))
print(resta(8,6))
print(multiplica(8,6))
```

modulo1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Modulo

Namespace

```
import modulo1
print( dir() )
```

['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'modulo1']

modulo1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Modulo

Modulo math

```
import math
print(dir(math))
```

['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'cbrt', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'exp2', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'sumprod', 'tan', 'tanh', 'tau', 'trunc', 'ulp']

Modulo

Modulo random

```
import random
print(dir(random))
```

['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_ONE', '_Sequence', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_accumulate', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_fabs', '_floor', '_index', '_inst', '_isfinite', '_lgamma', '_log', '_log2', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt', '_test', '_test_generator', '_urandom', '_warn', 'betavariate', 'binomialvariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', 'randbytes', 'randint', **'random'**, 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', **'uniform'**, 'vonmisesvariate', 'weibullvariate']

Modulo

Modulo random

```
import random as r
lista = ["Gustavo","Juana","Luis","Claudia"]
print("Lista:", lista )
print("Aleaotiro:", r.choice(lista))
r.shuffle(lista)
print("Lista mezclada:", lista )
```

Paquetes

¿Qué son los paquetes?

Directorios donde se almacenan módulos relacionados entre si.

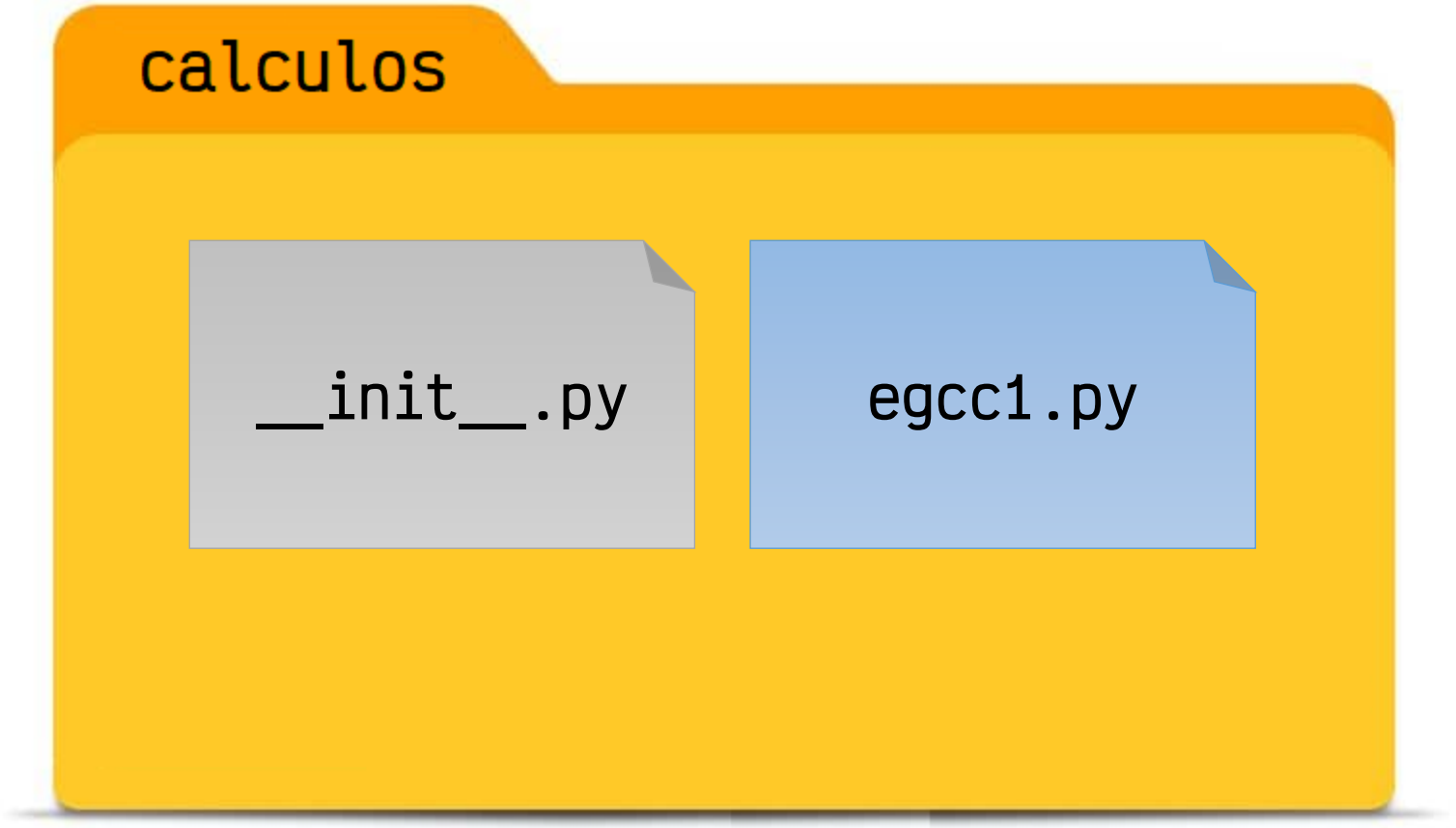
¿Para que sirven?

Para organizar y reutilizar los módulos.

¿Cómo se crea un paquete?

Se debe crear una carpeta con un archivo de nombre **__init__.py**

Paquetes



Paquetes

```
from calculos.egcc1 import *
print( suma(8,5) )
```

egcc1.py

```
def suma(num1, num2):
    rpta = num1 + num2
    return rpta

def resta(num1, num2):
    rpta = num1 - num2
    return rpta

def multiplica(num1, num2):
    rpta = num1 * num2
    return rpta
```

Excepciones

Eventos inesperados que ocurren durante la ejecución de un programa.

Las excepciones son instancias de una clase que se deriva de BaseException.

La clase BaseException es la clase base para todas las excepciones predefinidas en Python.

Cuando se produce una excepción, se puede manejar mediante una cláusula try-except.

```
try:
    print(10/0)
except:
    print("Error en la operación")
print("Fin del programa")
```

Excepciones

Es recomendable utilizar excepciones especificas según el tipo de error.

```
try:
    print(10/0)
except ZeroDivisionError:
    print("No se puede divir por cero.")
print("Fin del programa")
```

Evaluación continua

Desarrollar los problemas propuestos para esta semana


GRACIAS
TOTALES