~\Desktop\import itertools.py

```python
import itertools

# count: Genera una secuencia infinita que empieza en 10 y se incrementa de 2 en 2
print("count:")
for i in itertools.count(10, 2):
    print(i)
    if i >= 20:
        break

# cycle: Itera infinitamente sobre la secuencia proporcionada
print("\ncycle:")
counter = 0
for item in itertools.cycle('ABC'):
    print(item)
    counter += 1
    if counter >= 6:
        break

# chain: Une múltiples iteradores en uno solo
print("\nchain:")
for item in itertools.chain('ABC', 'DEF'):
    print(item)

# combinations: Genera todas las combinaciones posibles de una longitud especificada
print("\ncombinations:")
for combo in itertools.combinations('ABCD', 2):
    print(combo)

# permutations: Genera todas las permutaciones posibles de una longitud especificada
print("\npermutations:")
for perm in itertools.permutations('ABC', 2):
    print(perm)

from collections import deque, Counter, OrderedDict, defaultdict, namedtuple

# deque: Cola de doble extremo
print("\ndeque:")
d = deque([1, 2, 3])
d.append(4)
d.appendleft(0)
print(d)

# Counter: Cuenta elementos en una secuencia
print("\nCounter:")
c = Counter('abracadabra')
print(c)

# OrderedDict: Diccionario que mantiene el orden de inserción
print("\nOrderedDict:")
od = OrderedDict()
od['one'] = 1
```

```
od['two'] = 2
od['three'] = 3
print(od)

# defaultdict: Diccionario con un valor por defecto para claves inexistentes
print("\ndefaultdict:")
dd = defaultdict(int)
dd['one'] += 1
dd['two'] += 2
print(dd)

# namedtuple: Crea tuplas con nombre
print("\nnamedtuple:")
Point = namedtuple('Point', 'x y')
p = Point(10, 20)
print(p)
print(p.x, p.y)

import operator

# add: Suma dos números
print("add:")
result = operator.add(5, 3)
print(result)

# mul: Multiplica dos números
print("\nmul:")
result = operator.mul(5, 3)
print(result)

# itemgetter: Obtiene el valor de un índice específico en una secuencia
print("\nitemgetter:")
getter = operator.itemgetter(1)
sequence = [1, 2, 3]
print(getter(sequence))

# attrgetter: Obtiene el valor de un atributo específico en un objeto
print("\nattrgetter:")
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person('Alice', 30)
getter = operator.attrgetter('name')
print(getter(person))

from contextlib import contextmanager, closing, suppress
from urllib.request import urlopen

# contextmanager: Define un gestor de contexto
print("\ncontextmanager:")
@contextmanager
def simple_context_manager():
```

```
106     print("Entering")
107     yield
108     print("Exiting")
109 
110 with simple_context_manager():
111     print("Inside")
112 
113 # closing: Asegura que un recurso se cierre al finalizar
114 print("\nclosing:")
115 with closing(urlopen('http://www.python.org')) as page:
116     print(page.status)
117 
118 # suppress: Suprime excepciones específicas
119 print("\nsuppress:")
120 with suppress(FileNotFoundError):
121     open('nonexistentfile.txt')
122     print("File not found error suppressed")
123 
124 from functools import lru_cache, partial, reduce, wraps
125 
126 # lru_cache: Memoriza resultados de una función
127 print("\nlru_cache:")
128 @lru_cache(maxsize=None)
129 def fibonacci(n):
130     if n < 2:
131         return n
132     return fibonacci(n-1) + fibonacci(n-2)
133 
134 print(fibonacci(10))
135 
136 # partial: Crea una nueva función con argumentos parcialmente aplicados
137 print("\npartial:")
138 def multiply(x, y):
139     return x * y
140 
141 double = partial(multiply, 2)
142 print(double(5))
143 
144 # reduce: Aplica una función de manera acumulativa a los elementos de una secuencia
145 print("\nreduce:")
146 result = reduce(operator.add, [1, 2, 3, 4])
147 print(result)
148 
149 # wraps: Copia metadatos de una función a otra
150 print("\nwraps:")
151 def my_decorator(f):
152     @wraps(f)
153     def wrapper(*args, **kwargs):
154         print("Calling function")
155         return f(*args, **kwargs)
156     return wrapper
157 
158 @my_decorator
159 def say_hello():
```

```
160     print("Hello")
161
162 say_hello()
163
164 import math
165
166 # sqrt: Calcula la raíz cuadrada de un número
167 print("\nsqrt:")
168 print(math.sqrt(16))
169
170 # factorial: Calcula el factorial de un número
171 print("\nfactorial:")
172 print(math.factorial(5))
173
174 # gcd: Calcula el máximo común divisor de dos números
175 print("\ngcd:")
176 print(math.gcd(48, 18))
177
178 # log: Calcula el logaritmo natural de un número
179 print("\nlog:")
180 print(math.log(10))
181
182 import datetime
183
184 # date: Representa una fecha
185 print("\ndate:")
186 today = datetime.date.today()
187 print(today)
188
189 # time: Representa un tiempo
190 print("\ntime:")
191 now = datetime.datetime.now().time()
192 print(now)
193
194 # datetime: Representa una combinación de fecha y tiempo
195 print("\ndatetime:")
196 now = datetime.datetime.now()
197 print(now)
198
199 # timedelta: Representa una duración
200 print("\ntimedelta:")
201 delta = datetime.timedelta(days=5)
202 print(now + delta)
203
204 import datetime
205
206 # date: Representa una fecha
207 print("\ndate:")
208 today = datetime.date.today()
209 print(today)
210
211 # time: Representa un tiempo
212 print("\ntime:")
213 now = datetime.datetime.now().time()
```

```python
print(now)

# datetime: Representa una combinación de fecha y tiempo
print("\ndatetime:")
now = datetime.datetime.now()
print(now)

# timedelta: Representa una duración
print("\ntimedelta:")
delta = datetime.timedelta(days=5)
print(now + delta)

import re

# match: Busca una coincidencia al inicio de la cadena
print("\nmatch:")
match = re.match(r'\d+', '123abc')
print(match.group())

# search: Busca una coincidencia en toda la cadena
print("\nsearch:")
search = re.search(r'\d+', 'abc123')
print(search.group())

# findall: Encuentra todas las coincidencias en la cadena
print("\nfindall:")
findall = re.findall(r'\d+', 'abc123def456')
print(findall)

# sub: Reemplaza coincidencias en la cadena
print("\nsub:")
sub = re.sub(r'\d+', 'X', 'abc123def456')
print(sub)

import json

# load: Carga datos desde un archivo JSON
print("\nload:")
json_str = '{"name": "Alice", "age": 25}'
data = json.loads(json_str)
print(data)

# loads: Carga datos desde una cadena JSON
print("\nloads:")
json_str = '{"name": "Alice", "age": 25}'
data = json.loads(json_str)
print(data)

# dump: Escribe datos a un archivo JSON
print("\ndump:")
data = {'name': 'Alice', 'age': 25}
json_str = json.dumps(data)
print(json_str)

```

```python
# dumps: Convierte datos a una cadena JSON
print("\ndumps:")
data = {'name': 'Alice', 'age': 25}
json_str = json.dumps(data)
print(json_str)

import os

# listdir: Lista archivos en el directorio actual
print("\nlistdir:")
print(os.listdir('.'))

# mkdir: Crea un directorio
print("\nmkdir:")
os.mkdir('test_dir')
print('Directory created')

# remove: Elimina un archivo
print("\nremove:")
with open('test_file.txt', 'w') as f:
    f.write('test')
os.remove('test_file.txt')
print('File removed')

# path: Manipula rutas de archivos
print("\npath:")
print(os.path.join('dir', 'file.txt'))

import sys

# argv: Argumentos pasados al script desde la línea de comandos
print("\nargv:")
print(sys.argv)

# exit: Finaliza la ejecución del programa
print("\nexit:")
# sys.exit("Exiting program")

# path: Rutas de búsqueda de módulos
print("\npath:")
print(sys.path)

# stdout: Salida estándar
print("\nstdout:")
sys.stdout.write("Hello, stdout!\n")

import logging

# basicConfig: Configura el registro básico
print("\nbasicConfig:")
logging.basicConfig(level=logging.INFO)
logging.info('Este es un mensaje informativo')

# getLogger: Obtiene un logger
```

```
322 print("\ngetLogger:")
323 logger = logging.getLogger('example_logger')
324 logger.setLevel(logging.DEBUG)
325 logger.debug('Este es un mensaje de depuración')
326
327 # info: Registra un mensaje informativo
328 print("\ninfo:")
329 logger.info('Este es un mensaje informativo')
330
331 # error: Registra un mensaje de error
332 print("\nerror:")
333 logger.error('Este es un mensaje de error')
334
```