



EXPLORER


```
'''
percipio57_Simple Generators.py
Percipio video: Iterables-and-Generators; Simple Generators

* Demonstrate a simple generator
* Using 'yield' instead of 'return' makes a generator
* Simple generators can be functions that act like iterators where they return a value.  The
    value can be used from a previous invocation & used to return the next value.
* Like an iterator, the next() method or function can be used to display the values. Can also
    iterate over using a normal loop.

* Purpose of Generator expressions and generator objects is to avoid creating alot of objects
    kept in memory at once
'''
nl = '\n'
# crude form of a simple generator
def abc_generator(): # function defined as 'abc_generator'.
    yield 'a' # When this function is 1st called, it will yield 'a'.  it remembers 'a' is
        yielded & the position,...
    yield 'b' # ... so the next time this function is called, it will yield 'b',...
    yield 'c' # ... & so on until there are no more yields resulting in a StopIteration error
        or if iterating over in a loop (see below), then it will stop printing values.

for char in abc_generator(): # forLoop iterating over the yield values in the 'abc_generator'
    print(char) #
# more typical version of a simple generator
def num_generator(num=1): # number generator with a 'num' value = to 1
    while num: # as lomg as num == True, this while loop runs
        yield num # the number (num) is yielded
        num += 1 # 1st time through this loop yields 1, then the next time through, the loop
            remembers it has excuted the yield, it will increment by 1, check to see if 'num'
            is still True, & then will yield the next number, and so on.
        # This while loop can go on forever

for num in num_generator(): # forLoop used to iterate over the values that are generated by
    starting out with one number (num) in the 'num_generator'
    print(num) # prints out that number (num)
    if num == 5: # if the number equals 5 then is breaks the loop
        break #
```

EXPLORER

OPEN EDITORS 1 UNSAVED

- excel_code_.py
- python_debug_logging_code...
- percipio57_Simple Generator...
- percipio58_Lazy Generators....

PYTHON

- 02_Data-Sequence Types
- 03_Collections-Mapping-Loop...
- 04_Modules-Functions
- 05_Classes
- 06_Working-with-Files
- 07_Comprehensions
- 08_Iterables-and-Generators
  - percipio50_Basic Iteration.py
  - percipio51_The map() Functi...
  - percipio52_The Filter() Funct...
  - percipio53_The functools.re...
  - percipio54_Implementing an...
  - percipio55_Implement an Ite...
  - percipio56_Implement an Ite...
  - percipio57_Simple Generato...
  - percipio58_Lazy Generators....
  - percipio59_Recursive Genera...
  - percipio60_Exercise-Creating...
- 09_Exceptions
- 10_Automation Programming

Python Projects_2014

CMD_Python_Set-Path.txt

excel_code_.py

excel_code_summary_master

excel_code_summary_master.py

PIP_Help-2.PNG

PIP_Help.PNG

Python_Clear-Window-Comman...

python_debug_logging_code.py

python_exercises_00.py

python_exercises_01.py

Python_Tutorial_Running-Scripts...

Python_Tutorials.md

Scripts - Shortcut.lnk

start_code_for_python_master.py

excel_code_.py | python_debug_logging_code.py | percipio57_Simple Generators.py ● | percipio58_Lazy Generators.py

```
33  print(nl, 'Generator Syntax being used')
34  # Generator Syntax being used
35  # If there's an expression in parenthesis, that is used as a generator.
36  def doubles(stop=10): # while this function does not have a 'yield' statement, it does have a
        generator expression that it returns (like a list comprehension except doesn't use square
        brackets, uses parenthesis)
37      return (2 * n for n in range(stop)) # unlike a list comprehension which would generate
            every value in the list, this generator expression generates one value at a time on
            demand.
38
39  d_gen = doubles(5) # create an instance of doubles() function & passing the value 5
40  print('The object d_gen has type:', type(d_gen)) # prints type of object 'd_gen'
41  print('The first d_gen has type:', next(d_gen)) # using next, like on an iterator, view the
        next value generated
42  print('The second d_gen has type:', d_gen.__next__()) # use the special next method to obtain
        the next value or iteration
43
44  # Do not need to define a function to return a generator expression, as done above. Generator
        expressions can be used directly to create generator objects.
45  triples = (3 * n for n in range(10)) #  triples generates 3 * 'n' for each 'n' in the range to
        10
46  print('The object triples has type:', type(triples)) #
47  print('The first triples number type:', next(triples)) #
48  print('The second triples number type:', triples.__next__()) #
49  '''
50  RESULT:
51  a
52  b
53  c
54  1
55  2
56  3
57  4
58  5
59  The object d_gen has type: <class 'generator'>
60  The first d_gen has type: 0
61  The second d_gen has type: 2
62  The object triples has type: <class 'generator'>
63  The first triples number type: 0
64  The second triples number type: 3
65  '''
```