PYTHON

Mohan M J

Sequence

```
>>> # Fibonacci series:
# the sum of two elements
defines the next
... a, b = 0, 1
>>> while b < 10:
...     print(b)
...     a, b = b, a+b
```

```
>>> a, b = 0, 1
>>> while b < 1000:
...     print(b, end=',')
...     a, b = b, a+b
...
1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

if Statements

There can be zero or more elif parts, and the else part is optional.

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
```

for Statement

Python’s for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
--------------------------------------------------
>>> for w in words[:]:  # Loop over a slice copy
of the entire list.
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']
```

The range() Function

If you do need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates arithmetic progressions:

```
>>> for i in range(5):
...     print(i)
range(5, 10)                    5, 6, 7, 8, 9
range(0, 10, 3)                 0, 3, 6, 9
range(-10, -100, -30)          -10, -40, -70
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print(i, a[i])

>>> print(range(10))         range(0, 10)

>>> list(range(5))           [0, 1, 2, 3, 4]
```

Defining Functions

keyword def introduces a function definition.

create a function that writes the Fibonacci series to an arbitrary boundary

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print(a, end=' ')
...         a, b = b, a+b
>>> # Now call the function we just defined:
... fib(2000)
>>> fib                 <function fib at 10042ed0>
>>> f = fib
>>> f(100)
0 1 1 2 3 5 8 13 21 34 55 89
```

Defining Functions

write a function that returns a list of the numbers of the Fibonacci series, instead of printing it

```
>>> def fib2(n):  # return Fibonacci series up to n
...     """Return a list containing the Fibonacci series up to n."""
...     result = []
...     a, b = 0, 1
...     while a < n:
...         result.append(a)    # see below
...         a, b = b, a+b
...     return result
...
>>> f100 = fib2(100)    # call it
>>> f100                # write the result
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

pass Statements

Pass can be used when a statement is required syntactically but the program requires no action.

pass can be used is as a place-holder for a function or conditional body when you are working on new code, allowing you to keep thinking at a more abstract level. The pass is silently ignored

```
>>> while True:
...     pass  # Busy-wait for keyboard interrupt (Ctrl+C)
...

>>> class MyEmptyClass:
...     pass
...
```

break and continue Statements

The break statement, like in C, breaks out of the innermost enclosing for or while loop.

loop’s else clause runs when no break occurs.

Continue is used to end current iteration and “jump” to the beginning of the next.

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a
factor
...         print(n, 'is a prime number')

>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print("Found an even number", num)
...         continue
...     print("Found a number", num)
```

THANK YOU