Chapter 3: Loops

Last Update: January 31, 2023

Program 1: The program shows the two types of loops in their simplest form.

03 11-types of loops.py

```python
def next():
    input("\nNext> ")

for i in range(10):
    print(i, end=', ')
next()

i = 0
while i<10:
    print(i, end=', ')
    i += 1
```

Program 2: The purpose of this example is to show how an iterator works. There are easier ways to use a loop, as seen in Program 1. We are using a list as an example.

03 12-iterator.py

```python
# The next() function is NOT the same as the one we have been using.
# Initializing a list
mylist = ['apple', 'banana', 'cherry']
# Get the iterator
myit = iter(mylist)
# Verify the type of the iterator
print(type(myit))

# get the next item
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))   # what went wrong?
```

Program 3: Various examples of using range(). A range is not a list but can be cast into a list.

```
03  13-range.py
def next():
    input("next> ")

print(list(range(5)))
next()
print(list(range(1,5)))
next()
print(list(range(0,5,2)))
next()
print(list(range(0,10,3)))
next()
print(list(range(5,0,-1)))
next()
print(list(range(0,5,-2)))
```

Program 4:

```
03  14-loop and range.py
def next():
    input("next> ")

for i in range(0,5,2):
    print(i)
next()

for i in range(0,5,-2):
    print(i)
next()

for i in range(5,0,-2):
    print(i)
next()

start = int(input("Start = "))
stop  = int(input("Stop  = "))
for i in range(start, stop):
    print(i)
```

Program 5: Even though a string is similar to a list of characters, it is not. This program demonstrates how we loop through them. Look how similar they are. To save screen space, we prefer to print a list or a string horizontally instead of vertically.

03	15-range-string.py

```python
def next():
    input('\nNext> ')

s = 'Python'
print(f"string: {s}")
next()
# Loop over a string
for ch in s:
    print(ch, sep=' ', end=' ')
print()
next()

# Cast
slist = list(s)
print(f"list: {slist}")
next()
# Loop over a list of characters
for ch in slist:
    print(ch, sep=' ', end=' ')
print()
```

Program 6: This program shows how to print a range, a list of numbers, a string, and a list of strings. For printing a list, you can use an index or without one. Note that we use "I" for the index when there is one. Also, we tried to print the output using the same format, i. e., a space separating elements of the list elements. There is an important difference between the two methods (with the index or not); we will discuss that in the chapter about lists because it is a list issue, not a loop issue.

```
03  21-for.py
def next():
    input("\nnext > ")

print("1. Range/Index")
for i in range(10):
    print(i, end=' ')
next()

print("2. list-numbers")
for num in [11, 12, 13, 14, 15]:
    print(num, end=' ')
next()

print("3. string - with index")
word = 'Python 3.10'
for i in range(len(word)):
    print(word[i], end=' ')
next()

print("4. string - without index")
word = 'Python 3.10'
for letter in word:
    print(letter, end=' ')
next()

print("5. list of strings - without index")
fruits = ['banana', 'apple',  'mango']
for fruit in fruits:
    print (fruit, end=' ')
next()

print("6. list of strings - with index")
fruits = ['banana', 'apple',  'mango']
for i in range(len(fruits)):
    print (fruits[i], end=' ')
next()
```

Program 7: We are trying to abuse the for-loop in this program. Once the for-loop is set up, it is virtually not possible to change the setup. (You can use a break, though.)

03 22-index.py

```python
def next():
    input("\n\nNext > ")

n = 5
print("Case 1:  ")
for i in range(n):
    print(i, end=', ')
next()

print("Case 2:  ")
for i in range(n):
    print(i, end=', ')
    n = 7
# Changing n does not change the iteration of for
next()

print("Case 3:  ")
for i in range(n):
    print(i, end=', ')
    i = 2
# Even though i has been changed, it will be restored to the
# next value in the range
next()

print("Case 4:  ")
for i in range(n):
    i = 2
    print(i, end=', ')
next()
```

Program 8: When we use for-loop, we can obtain an index by calling len() and range() functions. We can avoid range() and len() by using an iterator, but we lose the index. Is there a way we can have the index without using range() and len()? This program shows us how to do it.

```
03  23-enumerate.py
```

```python
def next():
    input("\nnext > ")

fruits = ['apple', 'banana','cherry']

# with index, range(), len()
for i in range(len(fruits)):
    print(i, fruits[i])
next()

# without index, no range(), no len()
for fruit in fruits:
    print(fruit)
next()

# enumerate(), with index, no range(), no len()
for i, fruit in enumerate(fruits):
    print(i, fruit)
next()

for i, fruit in enumerate(fruits, 1):
    print(i, fruit)
```

Program 9: For-loops work well when we have a definite iteration. How about when we don't know the number of iterations beforehand? This is a place where we can use a "break".

```
03 24-for else.py
```

```python
def next():
    input("next > ")

list = [11, 22, 33, 44, 55, 66, 77]
#list = [11, 22, 33]
print(f"\n*** Case 1: break out")
sum = 0
for num in list:
    print(num)
    sum += num
    if sum > 100:
        print(f"The first sum >= 100 is {sum}")
        break
next()

print(f"\n*** Case 2: with else")
sum = 0
for num in list:
    print(num)
    sum += num
    if sum > 100:
        print(f"The first sum >= 100 is {sum}")
        break
else: # reaching the end, no break
    print("Not enough numbers")
next()

print(f"\n*** Test 3: without else")
sum = 0
for num in list:
    print(num)
    sum += num
    if sum > 100:
        print(f"The first sum >= 100 is {sum}")
        break
if sum < 100:
    print("Not enough numbers")
```

Program 10: This program provides another solution (without using for-else) for Case 1 of the last program.

```
03 25-break.py
list = [11, 22, 33]
print(f"\n*** Case 1: break out")
sum = 0
for num in list:
    print(num)
    sum += num
    if sum > 100:
        break
if sum <= 100:
    print("Not enough numbers")
else:
    print(f"The first sum >= 100 is {sum}")
```

Program 11: This program shows how to print a nice-looking list.

```
03 26-for end.py
```

```python
def next():
    input("\nNext > ")

n = 6
print('*** 1')
for i in range(n):
    print(i)
print("Make it Horizontal")
next()

print('*** 2')
for i in range(n):
    print(i, end=' ')
print("\nAdd commas.")
next()

print('*** 3')
for i in range(n):
    print(i, end=', ')
print()
print("Take out the extra comma.")
next()

print('*** 4')
for i in range(n):
    if i == n-1:
        end_str = '\n'
    else:
        end_str = ', '
    print(i, end=end_str)
print("make it shorter.")
next()

print('*** 5')
for i in range(n):
    if i == n-1:
        print(i, end = '\n')
    else:
        print(i, end = ', ')
print("Add a period at the end.")
next()

print('*** 6')
for i in range(n):
    print(i, end=', ' if i<n-1 else '. ')
print("\nThat looks perfect.")
```

Program 12: A simple example to compare for-loop with while-loop.

```
03 31-while vs for.py
idx = 0
while idx<10:
    print(idx, end=",   ")
    idx = idx + 1
print()

for idx in range(10):
    print(idx, end=",   ")
```

Program 13: There are many ways to stop a while loop.

```
03 32-sentinel.py
def next():
    input("\nNext> ")

print("Case 1: no sentinel")
sum = num = 0
size = int(input("How many numbers to add: "))
for i in range(size):
    value = int(input("Enter a positive int: "))
    sum = sum + value
    num = num + 1  # can be avoid
print("Sum of ", num, "numbers =", sum)
next()

print("Case 2: negative sentinel")
sum = num = 0
value = int(input("Enter a positive int: "))
while value > 0:
    sum = sum + value
    num = num + 1
    value=int(input("Enter a positive int: "))
print("Sum of ", num, "numbers =", sum)
next()

print("Case 3: while True")
sum = num = 0
while True:
    value = int(input("Enter a positive int: "))
    if value < 0:
        break
    else:
        sum = sum + value
        num = num + 1
print("Sum of ", num, "numbers =", sum)
next()

print("Case 4: switch")
sum = num = 0
done = False # We could use 'quit'
while not done:
    value = int(input("Enter a positive int: "))
    if value < 0:
        done = True
    else:
        sum = sum + value
        num += 1
else: # WILL NEVER REACH HERE
    print("*** On normal exit.")
print("Sum of ", num, "numbers =", sum)
```

Program 14: This program shows two slightly different ways to terminate a loop without using a break. Note that (a) we must make sure the Boolean expression controlling the loop is True, to begin with, and (b) the variable changes the value from True to False or the other way.

03	33-repeat or not.py

```python
def next():
    input("\nnext > ")

done = False
while not done:
    ch = input('Do you want to continue? ')
    if ch in 'yY':
        print('   As you wish ...')
        print("   Whatever you want to do here.")
    else:
        done = True
next()

repeat = True   # Too bad we can not use "continue"
while repeat:
    ch = input('Do you want to continue? ')
    if ch in 'yY':
        print('   As you wish ...')
        print("   Whatever you want to do here.")
    else:
        repeat = False
```

Program 15: Given an integer m, find the smallest n such that m >= 2**n.

03	34-power 2.py

```python
m = int(input("Enter a positive integer: "))
n = 0
pwr = 1
while pwr < m :
    pwr *= 2
    n += 1
    print(n, pwr)
print(f"{m} <= 2**{n}")
```

Program 16: Greatest Common Divisor (GCD).

```
03  35-GCD.py
num1 = int(input("Number 1: "))
num2 = int(input("Number 2: "))
print("The GCD of", num1, "and", num2)
while num1 != num2:
    if num1 < num2:
        num2 = num2 - num1
    else:
        num1 = num1 - num2
    print("\t", num1, num2) # For trace only
print ("is", num1)
```

Program 17: Trace a while loop. We added some print statements to show how the loop executes. The original code can be found on the slide.

```
03  30-trace.py
i, n = 1, 3
print(f"Output   i <= n   T/F")
print(f"------   ------   -----")
while i <= n:
    print(f"         {i} <= {n}  {i <= n}")
    print(i)
    i = i+1
print(f"         {i} <= {n}  {i <= n}    ")
```

Program 18: This program prints matrices of 3 x 5 and 5 x 3.

```
03  41a-nested loops.py
def next():
    input("next > ")

row, column = 3, 5
for i in range(row):
    for j in range(column):
        print('* ', sep='', end='')
    print()
next()

row, column = 5, 3
for i in range(row):
    for j in range(column):
        print('* ', sep='', end='')
    print()
```

Program 19: We are printing "triangular" matrices in this program. We are printing square matrices with two characters in the first two cases. The last two examples print "triangular."

03 41b-nested loops.py

```python
def next():
    input("next > ")

size = 6
for i in range(size):
    for j in range(size):
        print('* ' if j<=i else '- ', sep='', end='')
    print()
print()
next()

for i in range(size):
    for j in range(size):
        print('  ' if j<i else '* ', sep='', end='')
    print()
next()

for num_stars in range(0,size):
    for i in range(num_stars+1):
        print('* ', sep='', end='')
    print()
next()

for num_stars in range(size, 0, -1):
    for i in range(num_stars):
        print('* ', sep='', end='')
    print()
```

Program 20: The program produces a bar chart.

03 42-nested loops bar.py

```python
list_bars = [6, 8, 7, 1, 5, 9, 2]

for num in list_bars:
    print(num, ': ', sep='', end='')
    for i in range(num):
        print('*', sep='', end='')
    print()
```

Program 21: This program produces a 9 x 9 multiplication table.

03 43-nested table.py

```python
table_size = 10

print("\t\tMultiplication Table")
print('     ', end='')
for i in range (1,table_size):
    print(f'{i:4d}', end='')
print()
print('    ', "---+"*(table_size-1))

for i in range(1,table_size):
    print(f'{i:3d}:', end=' ')
    for j in range(1,table_size):
        print(f'{i*j:4d}', end="")
    print()
```

Program 22: This program shows several ways to print a list with or without enumeration.

03 44-enumerate.py

```python
def next():
    input("next> ")

fruits = ["apple", "banana", "cherry", "pear", "grape", "watermelon"]
print(fruits)
next()

print("\t Case 1: no index")
for fruit in fruits:
    print(fruit)
next()

print("\t Case 2: enumerate")
for idx, fruit in enumerate(fruits):
    print(idx, fruit)
next()

print("\t Case 3: enumerate starting with 3")
for num, fruit in enumerate(fruits, start=3):
    print(num, fruit)
next()

print("\t Case 4: iterator")
# from index to list element
for idx in range(len(fruits)):
    print(idx, fruits[idx])
```

Program 23: Using enumeration on a list.

```
03 45-enumerate list.py
def next():
    input("next> ")

colors = ["red", "green", "blue", "purple"]
ratios = [0.2, 0.3, 0.1, 0.4]

for elem in colors:
    print(elem)
next()

for elem in enumerate(colors):
    print(elem)
next()

for i, color in enumerate(colors, start=1):
    print(i, color)
next()

# loop over multiple iterables
# be careful not to use start=1!
for i, color in enumerate(colors, start=0):
    print(f"Color {i+1} {ratios[i]*100}% {color}")
```

Program 24: Using enumeration on a string.

```
03 46-enumeration string.py
str = "To be or not to be"
for i, ch in enumerate(str):
    print(f"{ch}, {i}")
```

Program 25: This program compares multiple ways to use a loop. The loop variable is a copy of the list element—more on this topic in the Chapter on Lists. If you use an index to access the list elements, you can change a list element. If you use an iterator, you won't be able to change the list element.

03 47-xxxx.py

```
def next():
    input("next > ")

fruits = ['apple', 'banana', 'cherry', 'pear']
print(fruits)

# Solution 1: While
i = 0
while i < len(fruits):
    if fruits[i]=="pear":
        fruits[i] = "watermelon"
    print(f"fruits[{i}] = {fruits[i]}")
    i += 1
print(fruits)
next()

# Solution 2: for index
fruits = ['apple', 'banana', 'cherry', 'pear']
print(fruits)
for i in range(len(fruits)):
    if fruits[i]=="pear":
        fruits[i] = "watermelon"
    print(f"fruits[{i}] = {fruits[i]}")
print(fruits)
next()

# Solution 3
fruits = ['apple', 'banana', 'cherry', 'pear']
print(fruits)
for fruit in fruits:
    if fruit=="pear":
        fruit = "watermelon"
    print(fruit)
print(fruits)
next()

# Solution 4: If you really want an index
fruits = ['apple', 'banana', 'cherry', 'pear']
print(fruits)
for i, fruit in enumerate(fruits):
    if fruit=="pear":
        fruits[i] = "watermelon"
    print(f"fruits[{i}] = {fruits[i]:10s}  {fruit:10s}")
print(fruits)
```

Program 26: The continue can skip some statements inside a loop. In the second loop, we used if-else to avoid using the continue.

```
03  51a-continue.py
# remove all "i" from a string
def next():
    input("next > ")

for ch in "string":
    if ch == "i":
        continue
    print(ch, end='')
else:
    print('\nNormal Exit')
print("The end")
next()

for ch in "string":
    if ch == "i":
        pass
    else:
        print(ch, end='')
else:
    print('\nNormal Exit')
print("The end")
```

Program 27: Practice using continue. In this program, we are trying to skip all vowels. We use a string to specify the five characters to skip.

```
03  51b-continue.py
# Try Texas, Mississippi, Washington
str = input("Enter a string: ")
for ch in str:
    if ch in "aeiou":
        continue
    print(ch, end='')
else:
    print('\nNormal Exit')
print("The end")
```

Program 28: Adding a break into the inner loop.

```
03 52a-break 2d.py
```

```python
def next():
    input("Next> ")

for i in range(8):
    for j in range(8):
        print("(", i, ",", j, ") ", sep='', end="")
    else:
        print()
else:
    print()
next()

for i in range(8):
    for j in range(8):
        print("(", i, ",", j, ") ", sep='', end="")
        if i == j:
            break
    print()
print()
```

Program 29: The first nested loop prints the indices of a 2d matrix. The second nested loop contains a break inside the inner loop. In the third case, we move the break to the outer loop. How does one break both inner and outer loops?

03 52b-break 2d.py

```python
def next():
    input("Next> ")

for i in range(8):
    for j in range(8):
        print("(", i, ",", j, ") ", sep='', end="")
    else:
        print()
next()

for i in range(8):
    for j in range(8):
        print("(", i, ",", j, ") ", sep='', end="")
        if i == 4:
            break
    print()
next()

for i in range(8):
    for j in range(8):
        print("(", i, ",", j, ") ", sep='', end="")
    if i == 4:
        break
    print()
print()
```

Program 30: This program tests if an integer is a prime number using a break.

```
03 53-break prime.py
# This is a simple code to test for prime numbers
import math
num = int(input("enter a positive number: "))
if num > 1:
    for i in range(2, math.ceil(math.sqrt(num))+1):
        print("Testing",i)
        if num%i == 0:
            print(f'*** {i} is a factor of {num}.')
            print(f'*** {num} is not a prime number.')
            break
    else:
        print(f'*** {num} is a prime number')
else:
    print(f'*** {num} is not a prime number.')
```

Program 31: The first loop uses a continue to control the loop, while the second loop uses a pass.

```
03 54-pass.py
def next():
    input("\n> ")

vowels = ['a', 'e', 'i', 'o', 'u']
test = input("Enter a string: ")

print(test, '>> ', end='')
for ch in test:
    if ch in vowels:
        continue
    print(ch, end='')
next()

print(test, '>> ', end='')
for ch in test:
    if ch in vowels:
        pass
    else:
        print(ch, end='')
```

Program 32: This program shows a general way to have a switch for exiting a while loop.

```
03  55-quit.py
```

```python
done = False
while not done:
    print("Do something here.")
    answer = input("Do you want to quit? ")
    if answer.lower().startswith("y"):
        done = True
print("All done.")
```

Program 33: While-else.

```
03  56-while else.py
```

```python
def next():
    input("Next> ")

i = 5
while (i <= 10):
    print (i)
    i = i + 1
    if i==7:
        break   # no the end
else: # only on normal exit
    print("End of the loop")
print("The next statement. \n")
next()

i = 5
while (i <= 10):
    print (i)
    i = i + 1
else:
    print("End of the loop")
print("The next statement. \n")
next()

while i < 3:
    print(i)
else:
    print('** end of the loop')
```