23.1 Loops (general)

Loop concept

People who have children may be familiar with looping around the block until a baby falls asleep.

PARTICIPATION ACTIVITY 23.1.1: Loop concept: Driving a baby around the block.

Animation captions:

1. Parents may be familiar with this scenario: Driving home, baby is awake. Parents circle the block, hoping the baby will fall asleep.
2. After first loop, baby is still awake, so parents loop again.
3. After second loop, baby is asleep, so parents head home for a peaceful evening.

PARTICIPATION ACTIVITY 23.1.2: Loop concept.

Consider the example above.

1) When the parents first checked, was the baby awake?
   - ○ Yes
   - ○ No

2) After the first loop, was the baby awake?
   - ○ Yes
   - ○ No

3) After the second loop, was the baby awake?
   - ○ Yes
   - ○ No

4) How many loops around the block did the parents make?
   - ○ 2

- 3

5) Where was the decision point for whether to loop: At the top of the street or bottom?

- Top
- Bottom

Loop basics

A ***loop*** is a program construct that repeatedly executes the loop's statements (known as the ***loop body***) while the loop's expression is true; when false, execution proceeds past the loop. Each time through a loop's statements is called an ***iteration***.

PARTICIPATION ACTIVITY 23.1.3: A simple loop: Summing the input values.

Animation captions:

1. A loop is like a branch, but jumping back to the expression when done. Thus, the loop's statements may execute multiple times, before execution proceeds past the loop.
2. This program gets an input value. If the value > -1, the program adds the value to a sum, gets another input, and repeats. val is 2, so the loop's statements execute, making sum 2.
3. The loop's statements ended by getting the next input, which is 4. The loop's expression 4 > -1 is true, so the loop's statements execute again, making sum 2 + 4 or 6.
4. The loop's statements got the next input of 1. The loop's expression 1 > -1 is true, so the loop's statements execute a third time, making sum 6 + 1 or 7.
5. The next input is -1. This time, -1 > -1 is false, so the loop is not entered. Instead, execution proceeds past the loop, where a statement puts sum, which is 7, to the output.

Loop example: Computing an average

A loop can be used to compute the average of a list of numbers.

PARTICIPATION ACTIVITY 23.1.4: Loop example: Computing an average.

Animation captions:

1. The program computes an average of a list of numbers (a negative ends the list). The first input is 2, so the loop is entered. Sum becomes 2, and num is incremented to 1.

2. The next input is 4. The loop is entered, so sum becomes 2 + 4 or 6, and num is incremented to 2.
3. The next input is 9, so the loop is entered. Sum becomes 6 + 9 or 15, and num is incremented to 3.
4. The next input is -1, so the loop is not entered. 15 / 3 or 5 is output.

PARTICIPATION ACTIVITY 23.1.5: Loop example: Average.

Consider the computing an average example above.

1) In the example above, the first value gotten from input was 2. That caused the loop body to be _____.
   - executed
   - not executed

2) At the end of the loop body, the _____.
   - next input is gotten
   - loop is exited
   - average is computed

3) With what value was sum initialized?
   - -1
   - 0

4) Each time through the loop, the sum variable is increased by _____.
   - 0
   - 1
   - the current input value

5) What was variable num's value after the loop was done iterating?
   - 1
   - 2
   - 3

6) Before the loop, the first input value is

gotten. If that input was negative (unlike the data in the example above), the loop's body would _____.

- be executed
- not be executed

Example: Counting specific values in a list

Programs execute one statement at a time. Thus, using a loop to examine a list of values one value at a time and updating variables along the way, as in the above examples, is a common programming task.

Below is a task to help a person get accustomed to examining a list of values one value at a time. The task asks a person to count the number of negative values, incrementing a variable to keep count.

PARTICIPATION ACTIVITY 23.1.6: Counting negative values in a list of values.

Click "Increment" if a negative value is seen.

Start

X	X	X	X	X	X	X

Counter

0

Next value Increment

Time - Best time -

Clear best

PARTICIPATION ACTIVITY 23.1.7: Counting negative values.

Complete the program such that variable count ends having the number of negative values in an input list of values (the list ends with 0). So if the input is -1 -5 9 3 0, then count should end with 2.

```
count = 0
val = Get next input

While val is not 0
   if __(A)__
      __(B)__

   val = Get next input
```


1) What should expression (A) be?

- val > 0
- val < 0
- val is 0

2) What should statement (B) be?

- val = val + 1
- count = count + 1
- count = val

3) If the input value is 0, does the loop body execute?

- Yes
- No

Example: Finding the max value

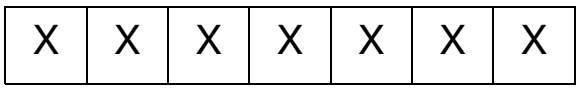
Examining items one at a time and updating a variable can achieve some interesting computations. The task below is to find the maximum value in a list of positive values. A variable stores the max value seen so far. Each input value is compared with that max, and if greater, that value replaces that max. The max value is initialized with -1 so that such comparison works even for the first input value.

PARTICIPATION ACTIVITY 23.1.8: Find the maximum value in the list of values.

Click "Store value" if a new maximum value is seen.

Start

max



Next value Store value

PARTICIPATION ACTIVITY 23.1.9: Determining the max value.

Complete the program such that variable max ends having the maximum value in an input list of positive values (the list ends with 0). So if the input is 22 5 99 3 0, then max should end as 99.

```
max = -1
val = Get next input

while val is not 0
   If __(A)__
      __(B)__

   val = Get next input
```

1) What should expression (A) be?

- max > 0
- max > val
- val > max

2) What should statement (B) be?

- max = val
- val = max
- max = max + 1

3) Does the final value of max depend on the order of inputs? In particular, would max be different for inputs 22 5 99 3 0 versus inputs 99 3 5 22 0?

- Yes
- No

4) For inputs 5 10 7 20 8 0, with what values should max be assigned?

- -1, 20

- -1, 5, 10, 20
- -1, 5, 10, 7, 20

23.2 More while examples

Example: GCD

The following is an example of using a loop to compute a mathematical quantity. The program computes the greatest common divisor (GCD) among two user-entered integers `num_a` and `num_b`, using Euclid's algorithm: If `num_a > num_b`, set `num_a` to `num_a - num_b`, else set `num_b` to `num_b - num_a`. Repeat until `num_a` equals `num_b`, at which point `num_a` and `num_b` both equal the GCD.

zyDE 23.2.1: While loop example: GCD program.

Try running the program below that calculates the greatest common divisor (GCD) of positive integers.

Load default template...

```
num_a = int(input('Enter first positive int
print()

num_b = int(input('Enter second positive in
print()

while num_a != num_b:
    if num_a > num_b:
        num_a = num_a - num_b
    else:
        num_b = num_b - num_a

print(f'GCD is {num_a}')

```

```
20
15
```

Run

PARTICIPATION ACTIVITY 23.2.1: Loop example: Greatest common divisor.

Use input values of `num_a = 15` and `num_b = 10` in the above GCD program. Try to

answer the questions by mentally executing the statements. If stuck, consider adding additional print statements to the program.

1) What is the value of num_a before the first loop iteration?

Check Show answer

2) What is num_a after the first and before the second iteration?

Check Show answer

3) What is num_b after the second and before the third iteration?

Check Show answer

4) How many loop iterations will the algorithm execute?

Check Show answer

Example: Conversation

Below is a program that has a "conversation" with the user. The program asks the user to type something and then randomly prints one of four possible responses until the user enters "Goodbye". Note that the first few lines of the program represent a ***docstring***: a multi-line string literal delimited at the beginning and end by triple-quotes. Either single ' or double " quotes can be used.

zyDE 23.2.2: While loop example: Conversation.

Run the program below. Try adding additional conditions to leave the conversation, s
"See you later". Program input must end with the string 'Goodbye'.

Load default ter


```
'''
Program that has a conversation with the user.
Uses elif branching and a random number to mix up the program's responses.
'''
import random  # Import a library to generate random numbers

print('Tell me something about yourself.')
print('You can type \'Goodbye\' at anytime to quit.\n')

user_text = input()

while user_text != 'Goodbye':
    random_num = random.randint(0, 2)  # Gives a random integer between 0 and
    if random_num == 0:
        print('\nPlease explain further.\n')
    elif random_num == 1:
        print(f"\nWhy do you say: '{user_text}'?\n")
```

```
Thank you very much, Mr. Robot.
Goodbye
```

Run

Each time through the while loop, the program will check if the user-entered string `user_text` is equal to the string literal "Goodbye". If the two strings are not equal, the while loop body executes. Within the while loop body, the program obtains a random number between 0 and 2 by using the random library. The ***randint()*** function provides a new random number each time the function is called. The arguments to randint(), 0 and 2, provide the minimum and maximum values that the function may return. Using the number given by randint(), the program's elif statements branch to a particular response.

PARTICIPATION ACTIVITY 23.2.2: Conversation program.

Refer to the above conversation program. If appropriate, type: unknown

1) Which if-else branch will execute if the user types "Goodbye"? Valid answers are branch 0, 1, 2 or none.

Check Show answer

2) How many times does the loop iterate in the program?

Check Show answer

3) Write an expression using random.randint() that returns a number between 0 and 5, inclusive.

Check Show answer

Example: Getting input until a sentinel is seen

Loops are commonly used to process a series of input values. A sentinel value is used to terminate a loop's processing. The example below computes the average of an input list of positive integers, ending with 0. The 0 is not included in the average.

©zyBooks 12/15/22 00:44 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

zyDE 23.2.3: Computing average of a list with a sentinel.

Load default ter

```
'''
Outputs average of list of positive integers
List ends with 0 (sentinel)
Ex: 10 1 6 3 0  yields (10 + 1 + 6 + 3) / 4, or 5
'''

values_sum = 0
num_values = 0

curr_value = int(input())

while curr_value > 0: # Get values until 0 (or less)
    values_sum += curr_value
    num_values += 1
    curr_value = int(input())

print(f'Average: {values_sum / num_values:.0f}\n')
```



```
10
1
6
```

Run

PARTICIPATION ACTIVITY 23.2.3: Average example with a sentinel.

Consider the example above and the given example input sequence `10 1 6 3 0`.

1) How many actual (non-sentinel) values are given in the first input sequence?

- 1
- 4
- 5

2) For the given input sequence, what is the final value of num_values?

- 0
- 4
- 5

3) Suppose the first input was 0. Would values_sum / num_values be 0?

- Yes
- No

4) What would happen if the following list was input: 10 1 6 3 -1?

- Output would be 5
- Output would be 4
- Error

CHALLENGE ACTIVITY 23.2.1: While loop with sentinel.

422102.2723990.qx3zqy7

Start

Type the program's output

```
curr_value = int(input())

minimum_number = curr_value

while curr_value > 0:
    if curr_value < minimum_number:
        minimum_number = curr_value
    curr_value = int(input())

print('Min value:', minimum_number, end='')
```

Input

```
44
23
56
12
0
```

Output

```
Min value: 1
```


1 2

Check Next

CHALLENGE ACTIVITY 23.2.2: Bidding example.

Write an expression that continues to bid until the user enters 'n'.

Sample output with inputs: 'y' 'y' 'n'

```
I'll bid $7!
Continue bidding? I'll bid $15!
Continue bidding? I'll bid $23!
Continue bidding?
```


422102.2723990.qx3zqy7

```
1  import random
2  random.seed(5)
3
4  keep_bidding = '-'
5  next_bid = 0
6
7  while ''' Your solution goes here ''':
8     next_bid = next_bid + random.randint(1, 10)
9     print(f'I\'ll bid ${next_bid}!')
10    print('Continue bidding?', end=' ')
11    keep_bidding = input()
```

Run

CHALLENGE ACTIVITY 23.2.3: While loop: Insect growth.

Given positive integer num_insects, write a while loop that prints, then doubles, num_insects each iteration. Print values ≤ 100. Follow each number with a space.

Sample output with input: 8

```
8 16 32 64
```

422102.2723990.qx3zqy7

```
1  num_insects = int(input()) # Must be >= 1
```

Run

23.3 Counting

Counting with a while loop

Commonly, a loop should iterate a specific number of times, such as 10 times. The programmer can use a variable to count the number of iterations, called a ***loop variable***. To iterate N times using an integer loop variable i, a loop[1] with the following form is used:

Construct 23.3.1: Counting while loop form.

```
# Iterating N times using a loop
variable
i = 1
while i <= N:
    # Loop body statements go here
    i = i + 1
```


A common error is to forget to include the loop variable update (e.g., i = i +1) at the end of the loop, causing an unintended infinite loop.

The following program outputs the amount of money in a savings account each year for the user-entered number of years, with $10,000 initial savings and 5% yearly interest:

zyDE 23.3.1: While loop that counts iterations: Savings interest program.

Load default ter

```
'''Program that calculates savings and interest'''

initial_savings = 10000
interest_rate = 0.05

print(f'Initial savings of ${initial_savings}')
print(f'at {interest_rate*100:.0f}% yearly interest.\n')

years = int(input('Enter years: '))
print()

savings = initial_savings
i = 1  # Loop variable
while i <= years:  # Loop condition
    print(f' Savings at beginning of year {i}: ${savings:.2f}')
    savings = savings + (savings*interest_rate)
    i = i + 1  # Increment loop variable
```


10

Run

PARTICIPATION ACTIVITY 23.3.1: Savings interest program.

Refer to the program above.

1) With an initial savings of $10000 and an interest rate of 0.05, what's the amount of savings at the beginning of year 4? Ignore cents and do not include the dollar sign ($).

Check **Show answer**

2) If the interest_rate is 3% and initial_savings are $5000, savings will be greater than $7500 after how many loop iterations?

Check Show answer

PARTICIPATION ACTIVITY 23.3.2: Basic counting with while loops.

Use <= in each answer.

1) Loop iterates 10 times.

```
i = 1
while
    :
    # Loop body statements go here
    i = i + 1
```

Check Show answer

2) Loop iterates 99 times.

```
i = 1
while
    :
    # Loop body statements go here
    i = i + 1
```

Check Show answer

3) Loop iterates 2 times.


```
i = 1
while
    :
    # Loop body statements go here
    i = i + 1
```

Check Show answer

Other forms of counting

Counting down is also common, as in counting down from 5 to 1. The loop body executes when i is 5, 4, 3, 2, and 1, but does not execute when i reaches 0.

Figure 23.3.1: While loop with loop variable that counts down.


```
i = 5
while i >= 1:
    # Loop body statements go
here
    i = i - 1
```

The loop body executes when i is 5, 4, 3, 2, and 1, but does not execute when i reaches 0.

Counting sometimes occurs by steps greater than 1. Ex: A loop that prints even values from 0 to 100 (i.e., counts from 0 to 100 by 2s) is:

Figure 23.3.2: Loop variable increased by 2 per iteration.

```
i = 0
while i <= 100:
    # Loop body statements go
here
    i = i + 2
```

zyDE 23.3.2: Loop over presidential election years.

Write a program that prints the U.S. presidential election years from 1792 to present (
knowing that such elections occur every 4 years.

Hint: Initialize your loop variable to 1792. Don't forget to use <= rather than == to help
infinite loop.

Load default template... **Run**

```
1 year = 1792
2 current_year = ?
3
4 while year ? ??:
5     # Print the election year
6     year = year + ?
7
```

PARTICIPATION ACTIVITY 23.3.3: Forms of counting.

Complete the missing parts of the code.

1) Loop iterates over the odd integers from 1 to 9 (inclusive).

```
i = 1
while i <= 9:
    # Loop body statements
go here
```


Check **Show answer**

2) Loop iterates over multiples of 5 from 0 to 1000 (inclusive).

```
i = 0
while
                    :
    # Loop body statements
go here
    i = i + 5
```

Check Show answer

3) [illegible]es over the odd integers from 211 down to 31 (inclusive).


```
i = 211
while i >= 31:
    # Loop body statements
go here
```

Check Show answer

4) Loop iterates from -100 to 65.

```
while i <= 65:
    # Loop body statements
go here
    i = i + 1
```

Check Show answer

PARTICIPATION ACTIVITY 23.3.4: Counting in a loop simulator.

The following tool allows you to enter values for a loop's parts, and then executes the loop. Using the tool, try to solve each listed problem individually.

The tool can use any relational or equality operator, such as <, <=, >, >=, ==, etc. Identity and membership operators like "is" or "in" will not work.

1. 0 to 10,000 by 500s (0, 500, 1000, ...)
2. -19 to 19 by 1s
3. 10 to -10 by 1s
4. Multiples of 3 between 0 and 100
5. Powers of 2 from 1 to 256 (1, 2, 4, 8, ...)
6. Come up with your own challenges

```
i =
while i :
   print(i, end=' ')
   i = i
```

Run code

Output is: `Awaiting your input...`

zyDE 23.3.3: Calculate a factorial.

Write a program that lets a user enter N and that outputs N! (N factorial, meaning N*(N-1)*(N-2)*...*2*1). Hint:Use a loop variable i that counts from total-1 down to 1. Compare output with some of these answers: 1:1, 2:2, 3:6, 4:24, 5:120, 8:40320.

Load default template...

```
1  N = int(input())  # Read user-entered numb
2  total = N
3  # Initialize the loop variable
4
5  while i ? ??:
6      # Set total to total * (i)
7      # Decrement i
8
9  print(total)
10
```

5

Run

Shorthand operators

Because assignments such as `i = i + 1` are so common in programs, the programming language provides a shorthand version: `i += 1`. Similar operators include +=, -=, *=, and /=. For example, `num *= x` is shorthand for `num = num * x`. The item on the right can be an expression, so `num *= x + y` is shorthand for `num = num * (x + y)`. Usage of such operators is common in loops.

Construct 23.3.2: Operators like += are common in loops.

```
i = 0
while i < N:
    # Loop body statements go
here
    i += 1
```


PARTICIPATION ACTIVITY 23.3.5: Shorthand operators.

Answer each question using the operators of the form +=, *=, /=, -=, etc.

1) Write a statement using *= that doubles the value of a variable my_var.

Check Show answer

2) Write a statement using += that is equivalent to `my_var = my_var + my_var / 2`

Check Show answer

CHALLENGE ACTIVITY 23.3.1: Loops with variables that count.

422102.2723990.qx3zqy7

Start

Type the program's output

```
n = 1
while n <= 6:
    print(n * 2)
    n = n + 1
```

```
2
4
6
8
10
12
```


1 | 2 | 3

CHALLENGE ACTIVITY 23.3.2: While loop: Print 1 to N.

Write a while loop that prints from 1 to user_num, increasing by 1 each time.

Sample output with input: 4

```
1
2
3
4
```

422102.2723990.qx3zqy7

```
1 i = 1
2
3 user_num = int(input()) # Assume positive
4
5 ''' Your solution goes here '''
6
```

Run

CHALLENGE ACTIVITY 23.3.3: Printing output using a counter.

Retype or copy, and then run the following code; note incorrect behavior. Then fix errors in the code, which should print num_stars asterisks.


```
while num_printed != num_stars:
    print('*')
```

Sample output with input: 3

```
*
*
*
```

422102.2723990.qx3zqy7

```
1 num_printed = 0
2
3 num_stars = int(input())
4
5 ''' Your solution goes here '''
6
```

Run

(*1) Focus is placed on mastering basic looping using while loops, before introducing for loops and range().

23.4 While vs. for loops

While loop and for loop correspondence

Both for loops and while loops can be used to count a specific number of loop iterations. A for loop combined with range() is generally preferred over while loops, since for loops are less likely to become stuck in an infinite loop situation. A programmer may easily forget to increment a while loop's variable (causing an infinite loop), but for loops iterate over a finite number of elements in a container and are thus guaranteed to complete.

PARTICIPATION ACTIVITY 23.4.1: While/for loop correspondence.

Animation captions:

1. A for loop and range function can replace a while loop.
2. A more concise for loop uses a single argument for range().

As a general rule:

1. *Use a for loop* when the number of iterations is computable before entering the loop, as when counting down from X to 0, printing a string N times, etc.
2. *Use a for loop* when accessing the elements of a container, as when adding 1 to every element in a list, or printing the key of every entry in a dict, etc.
3. *Use a while loop* when the number of iterations is not computable before entering the loop, as when iterating until a user enters a particular character.

These are not hard rules, just general guidelines.

PARTICIPATION ACTIVITY 23.4.2: While loops and for loops.

Indicate whether a while loop or for loop should be used in the following scenarios:

1) Iterate as long as the user-entered string c is not q.

- ○ while
- ○ for

2) Iterate until the values of x and y are equal, where x and y are changed in the loop body.

○ while

3) Iterate 1500 times

○ for
○ while

○ for

23.5 Nested loops

Nested loops

A ***nested loop*** is a loop that appears as part of the body of another loop. The nested loops are commonly referred to as the ***outer loop*** and ***inner loop***.

Nested loops have various uses. One use is to generate all combinations of some items. Ex: The following program generates all two letter .com Internet domain names. Recall that `ord()` converts a 1-character string into an integer, and `chr()` converts an integer into a character. Thus, `chr(ord('a') + 1)` results in 'b'.

Figure 23.5.1: Nested loops example: Two-letter domain name printing program.

```
"""
Program to print all 2-letter domain names.

Note that ord() and chr() convert between text and
the ASCII or Unicode encoding:
-  ord('a') yields the encoded value of 'a', the
number 97.
-  ord('a')+1 adds 1 to the encoded value of 'a',
giving 98.
-  chr(ord('a')+1) converts 98 back into a letter,
producing 'b'
"""
print('Two-letter domain names:')

letter1 = 'a'
letter2 = '?'
while letter1 <= 'z':  # Outer loop
    letter2 = 'a'
    while letter2 <= 'z':  # Inner loop
        print(f'{letter1}{letter2}.com')
        letter2 = chr(ord(letter2) + 1)
    letter1 = chr(ord(letter1) + 1)
```

```
Two-letter
domain names:
aa.com
ab.com
ac.com
ad.com
ae.com
af.com
ag.com
ah.com
ai.com
aj.com
ak.com
al.com
am.com
an.com
ao.com
ap.com
aq.com
ar.com
as.com
at.com
au.com
av.com
aw.com
ax.com
ay.com
az.com
ba.com
bb.com
...
zy.com
zz.com
```


(Forget about buying a two-letter domain name: They are all taken, and each sells for several hundred thousand or millions of dollars. Source: dnjournal.com, 2012.)

©zyBooks 12/15/22 00:44 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

zyDE 23.5.1: Two character dotcom domain names.

Modify the program to include two-character .com names, where the second charact a letter or a number, e.g., a2.com. Hint: Add a second while loop nested in the outer lo following the first inner loop, that iterates through the numbers 0-9.

Load default template... Run


```
'''
Program to print all 2-letter domain names
Note that ord() and chr() convert between t
ord('a') is 97. ord('b') is 98, and so on.
'''
print('Two-letter domain names:')

letter1 = 'a'
letter2 = '?'
while letter1 <= 'z':  # Outer loop
    letter2 = 'a'
    while letter2 <= 'z':  # Inner loop
        print(f'{letter1}{letter2}.com')
        letter2 = chr(ord(letter2) + 1)
    letter1 = chr(ord(letter1) + 1)

```

©zyBooks 12/15/22 00:44 1361995
John Farrell
COLOSTATECS220SeaboltFall2022

zyDE 23.5.2: Nested loop example: Histogram.

Here is a nested loop example that graphically depicts an integer's magnitude by usir asterisks, creating what is commonly called a *histogram*:

Run the program below and observe the output. Modify the program to print one aste units. So if the user enters 40, print 8 asterisks.

Load default template...

```
num = 0
while num >= 0:
    num = int(input('Enter an integer (nega
    if num >= 0:
        print('Depicted graphically:')
        for i in range(num):
            print('*', end=' ')
        print('\n')

print('Goodbye.')
```

```
5
10
-1
```

Run

PARTICIPATION ACTIVITY 23.5.1: Nested loops.

1) Given the following code, how many times will the inner loop body execute?

```
for i in range(2):
    for j in range(3):
        # Inner loop body
```

Check Show answer

2) Given the following code, how many times will the print statement execute?

```
for i in range(5):
    for j in range(10,
12):
                        }')
```

Check Show answer

3) What is the output of the following code?


```
c1 = 'a'
while c1 < 'b':
    c2 = 'a'
    while c2 <= 'c':
        print(f'{c1}
{c2}', end = ' ')
        c2 = chr(ord(c2)
+ 1)
    c1 = chr(ord(c1) + 1)
```

Check Show answer

4) What is the output of the following code?

```
i1 = 1
while i1 < 19:
    i2 = 3
    while i2 <= 9:
        print(f'{i1}
{i2}', end=' ')
        i2 = i2 + 3
    i1 = i1 + 10
```

Check Show answer

CHALLENGE ACTIVITY 23.5.1: Nested loops.

422102.2723990.qx3zqy7

Start

Type the program's output

```
count = 0
for i in range(3):
   for j in range(4):
      count = count + 1
print(count)
```

12

1	2	3	4

Check Next

CHALLENGE ACTIVITY 23.5.2: Nested loops: Print rectangle

Given the number of rows and the number of columns, write nested loops to print a rectangle.

Sample output with inputs: 2 3

```
* * *
* * *
```

422102.2723990.qx3zqy7

```
1 num_rows = int(input())
2 num_cols = int(input())
3
4 ''' Your solution goes here '''
5
6         print('*', end=' ')
7     print()
```


Run

CHALLENGE ACTIVITY 23.5.3: Nested loops: Print seats.

Given num_rows and num_cols, print a list of all seats in a theater. Rows are numbered, columns lettered, as in 1A or 3E. Print a space after each seat.

Sample output with inputs: 2 3

```
1A 1B 1C 2A 2B 2C
```


422102.2723990.qx3zqy7

```
num_rows = int(input())
num_cols = int(input())

# Note 1: You will need to declare more variables
# Note 2: Place end=' ' at the end of your print statement to separate seats by spaces

''' Your solution goes here '''

print()
```

Run

23.6 Developing programs incrementally

Incremental programming

As programs increase in complexity, a programmer's development process becomes more important. A programmer should not write the entire program and then run the program—hoping the program works. If, as is often the case, the program does not work on the first try, debugging at that point can be extra difficult because the program may have many distinct bugs.

Experienced programmers practice ***incremental programming***, starting with a simple version of the program, and then growing the program little-by-

little into a complete version.

Example: Phone number program

The following program allows the user to enter a phone number that includes letters, which appear on phone keypads along with numbers and are commonly used by companies as a marketing tactic (e.g., 1-555-HOLIDAY). The program then outputs the phone number using numbers only.

The first program version simply prints each element of the string to ensure the loop iterates properly through each string element.

Figure 23.6.1: First version echoes input phone number string.

```
user_input = input('Enter phone number:
')

index = 0
for character in user_input:
    print(f'Element {index} is:
{character}')
    index += 1
```

```
Enter phone number: 1-555-
HOLIDAY
Element 0 is: 1
Element 1 is: -
Element 2 is: 5
Element 3 is: 5
Element 4 is: 5
Element 5 is: -
Element 6 is: H
Element 7 is: O
Element 8 is: L
Element 9 is: I
Element 10 is: D
Element 11 is: A
Element 12 is: Y
```

The second program version outputs the numbers (0 - 9) of the phone number and outputs a '?' for all other characters. A ***FIXME comment*** attracts attention to code that needs to be fixed in the future. Many editors automatically highlight FIXME comments. Large projects with multiple programmers might also include a username and date, as in `FIXME(01/22/2018, John)`.

Figure 23.6.2: Second version echoes numbers, and has FIXME comment.

```
user_input = input('Enter phone number: ')
phone_number = ''

for character in user_input:
    if '0' <= character <= '9':
        phone_number += character
    else:
        #FIXME: Add elif branches for
letters and hyphen
        phone_number += '?'

print(f'Numbers only: {phone_number}')
```



```
Enter phone number: 1-555-
HOLIDAY
Numbers only: 1?555????????
```

The third version completes the elif branch for the letters A-C (lowercase and uppercase, per a standard phone keypad). The code also modifies the if branch to echo a hyphen in addition to numbers. The third version adds the elif branch for the letters A-C (lowercase and uppercase, per a standard phone keypad).

Figure 23.6.3: Third version echoes hyphens too, and handles first three letters.

```
user_input = input('Enter phone number: ')
phone_number = ''

for character in user_input:
    if ('0' <= character <= '9') or (character
== '-'):
        phone_number += character
    elif ('a' <= character <= 'c') or ('A' <=
character <= 'C'):
        phone_number += '2'
    #FIXME: Add remaining elif branches
    else:
        phone_number += '?'

print(f'Numbers only: {phone_number}')
```

```
Enter phone number:
1-555-HOLIDAY
Numbers only:
1-555-??????2
```


The fourth version can be created by filling in the elif branches similarly for other letters and adding

more instructions for handling unexpected characters. The code is not shown below, but sample input/output is provided.

Figure 23.6.4: Fourth and final version sample input/output.

```
Enter phone number (letters/- OK, no spaces): 1-555-HOLIDAY
Numbers only: 1-555-4654329
...
Enter phone number (letters/- OK, no spaces): 1-555-holiday
Numbers only: 1-555-4654329
...
Enter phone number (letters/- OK, no spaces): 999-9999
Numbers only: 999-9999
...
Enter phone number (letters/- OK, no spaces): 9876zywx%$#@
Numbers only: 98769999????
```


zyDE 23.6.1: Complete the phone number program.

Complete the program by providing the additional branches for decoding other letters phone number. Try incrementally writing the program by adding one elif branch at a t testing that each added branch works as intended.

Load default template...

```
1  user_input = input('Enter phone number:\n')
2  phone_number = ''
3
4  for character in user_input:
5      if ('0' <= character <= '9') or (charac
6          phone_number += character
7      elif ('a' <= character <= 'c') or ('A'
8          phone_number += '2'
9      #FIXME: Add remaining elif branches
10     else:
11         phone_number += '?'
12
13 print(f'Numbers only: {phone_number}')
14
```

1-555-HOLIDAY

Run

PARTICIPATION ACTIVITY 23.6.1: Incremental programming.

1) Incremental programming may help reduce the number of errors in a program.
   - True
   - False

2) FIXME comments provide a way for a programmer to remember what needs to be added.
   - True
   - False

3) Once a program is complete, one would expect to see several FIXME comments.
   - True
   - False

23.7 Break and continue

Break statements

A ***break*** statement in a loop causes the loop to exit immediately. A break statement can sometimes yield a loop that is easier to understand.

In the example below, the nested for loops generate possible meal options for the number of empanadas and tacos that can be purchased. The inner loop body calculates the cost of the current meal option. If the meal cost is equal to the user's amount of money, the search is over, so the break statement immediately exits the inner loop. The outer loop body also checks if the meal cost and the user's amount of money are equal, and if so, that break statement exits the outer loop.

The program could be written without break statements, but the loops' condition expressions would be more complex and the program would require additional code, perhaps being harder to understand.

Figure 23.7.1: Break statement.

```
empanada_cost = 3
taco_cost = 4

user_money = int(input('Enter money for meal: '))

max_empanadas = user_money // empanada_cost
max_tacos = user_money // taco_cost

meal_cost = 0
for num_tacos in range(max_tacos + 1):
    for num_empanadas in range(max_empanadas + 1):
        meal_cost = (num_empanadas * empanada_cost) + (num_tacos *
taco_cost)

        # Find first meal option that exactly matches user money
        if meal_cost == user_money:
            break

    # Find first meal option that exactly matches user money
    if meal_cost == user_money:
        break

if meal_cost == user_money:
    print(f'${meal_cost} buys {num_empanadas} empanadas and {num_tacos}
tacos without change.')
else:
    print('You cannot buy a meal without having change left over.')
```

```
Enter money for meal: 20
$20 buys 4 empanadas and 2 tacos without change.
...
Enter money for meal: 31
$31 buys 9 empanadas and 1 tacos without change.
```


PARTICIPATION ACTIVITY 23.7.1: Break statements.

Given the following while loop, what is the value variable z is assigned with for the given values of variables a, b and c?


```
mult = 0
while a < 10:
    mult = b * a
    if mult > c:
        break
    a = a + 1
z = a
```

1) a = 1, b = 1, c = 0

Check Show answer

2) a = 4, b = 5, c = 20

Check Show answer

Continue statements

A ***continue*** statement in a loop causes an immediate jump to the while or for loop header statement. A continue statement can improve the readability of a loop. The example below extends the previous meal finder program to find meal options for which the total number of items purchased is evenly divisible by the number of diners. In addition, the following program will output all possible meal options, instead of reporting the first meal option found.

The program uses two nested for loops to try all possible combinations of tacos and empanadas. If the total number of tacos and empanadas is not exactly divisible by the number of diners (e.g., `num_tacos + num_empanadas) % num_diners != 0`, the continue statement will immediately proceed to the next iteration of the for loop.

Break and continue statements can be helpful to avoid excessive indenting/nesting within a loop. However, because someone reading a program could easily overlook a break or continue statement, such statements should be used only when their use is clear to the reader.

Figure 23.7.2: Continue statement.

```
empanada_cost = 3
taco_cost = 4

user_money = int(input('Enter money for meal: '))

num_diners = int(input('How many people are eating: '))

max_empanadas = user_money // empanada_cost
max_tacos = user_money // taco_cost

meal_cost = 0
num_options = 0
for num_tacos in range(max_tacos + 1):
    for num_empanadas in range(max_empanadas + 1):

        # Total items purchased must be equally divisible by number of
diners
        if (num_tacos + num_empanadas) % num_diners != 0:
            continue

        meal_cost = (num_empanadas * empanada_cost) + (num_tacos *
taco_cost)

        if meal_cost == user_money:
            print(f'${meal_cost} buys {num_empanadas} empanadas and
{num_tacos} tacos without change.')
            num_options += 1

if num_options == 0:
    print('You cannot buy a meal without having change left over.')
```

```
Enter money for meal: 60
How many people are eating: 3
$60 buys 12 empanadas and 6 tacos without change.
$60 buys 0 empanadas and 15 tacos without change.
...
Enter money for meal: 54
How many people are eating: 2
$54 buys 18 empanadas and 0 tacos without change.
$54 buys 10 empanadas and 6 tacos without change.
$54 buys 2 empanadas and 12 tacos without change.
```


PARTICIPATION ACTIVITY 23.7.2: Continue statements.

Given:

```
for i in range(5):
    if i < 10:
        continue
    print(i)
```

1) The loop will print at least some output.

- True
- False

2) The loop will iterate only once.

- True
- False

CHALLENGE ACTIVITY 23.7.1: Enter the output of break and continue.

422102.2723990.qx3zqy7

Start

Type the program's output

```
stop = int(input())
result = 0
for n in range(10):
    result += n + 3
    if result > stop:
        break
    print(n)
print(result)
```

Input

8

Output

1 2 3 4

Check Next

CHALLENGE ACTIVITY 23.7.2: Simon says.

"Simon Says" is a memory game where "Simon" outputs a sequence of 10 characters (R, G, B, Y) and the user must repeat the sequence. Create a for loop that compares each character of the two strings. For each matching character, add one point to user_score. Upon a mismatch, end the loop.

Sample output with inputs: 'RRGBRYYBGY' 'RRGBBRYBGY'

```
User score: 4
```


422102.2723990.qx3zqy7

```
1 user_score = 0
2 simon_pattern = input()
3 user_pattern  = input()
4
5 ''' Your solution goes here '''
6
7 print('User score:', user_score)
```

Run

23.8 Loop else

Loop else construct

A loop may optionally include an else clause that executes only if the loop terminates normally, not using a break statement. Thus, the complete forms of while and for loops are:

Construct 23.8.1: While loop else.

```
while expression:  # Loop expression
    # Loop body: Sub-statements to execute if
    # the expression evaluated to True
else:
    # Else body: Sub-statements to execute
once
    # if the expression evaluated to False

# Statements to execute after the loop
```


Construct 23.8.2: For loop else.

```
for name in iterable:
    # Loop body: Sub-statements to
execute
    # for each item in iterable
else:
    # Else body: Sub-statements to
execute
    # once when loop completes

# Statements to execute after the loop
```

The ***loop else*** construct executes if the loop completes normally. In the following example, a special message "All names printed" is displayed if the entire list of names is completely iterated through.

Figure 23.8.1: Loop else branch taken if loop completes normally.

```
names = ['Janice', 'Clarice', 'Martin',
'Veronica', 'Jason']

num = int(input('Enter number of names to
print: '))
for i in range(len(names)):
    if i == num:
        break
    print(names[i], end= ' ')
else:
    print('All names printed.')
```

```
Enter number of names to
print: 2
Janice Clarice
...
Enter number of names to
print: 8
Janice Clarice Martin
Veronica Jason
All names printed.
```

zyDE 23.8.1: Loop else example: Finding a legal baby name.

The country of Denmark allows parents to pick from around 7,000 names for newbor Names not on the list must receive special approval from the Names Investigation Department of Copenhagen University. (Surprisingly, many countries have naming la probably to avoid names like "Brfxxccxxmnpccccllllmmnprxvclmnckssqlbb11116", pr "Albin".)

The program below checks if a user-entered name is an appropriate Danish name. If is not found in the list of legal names, then a suggestion is made to a close match. A match is an acceptable name starting with the same letter. If no close matches are f loop else clause informs the user. If there are multiple names with the same letter, th the list is used.

Run the program below.

1. Enter the acceptable name "Bjork".
2. Try the name "Michaal", which is not an acceptable name. The program will sug replacement since there is an acceptable name starting with 'M'.
3. Try the name "Zoidberg", which is not an acceptable name. The list doesn't cont name starting with 'Z', so the program will print a special message and termina

main.py Load default ter

```
#A few legal, acceptable Danish names
legal_names = ['Thor', 'Bjork', 'Bailey', 'Anders', 'Bent', 'Bjarne', 'Bjorn',
    'Claus', 'Emil', 'Finn', 'Jakob', 'Karen', 'Julie', 'Johanne', 'Anna', 'An
    'Bente', 'Eva', 'Helene', 'Ida', 'Inge', 'Susanne', 'Sofie', 'Rikkie', 'Pi
    'Torben', 'Soren', 'Rune', 'Rasmus', 'Per', 'Michael', 'Mads', 'Hanne',
    'Dorte'
]

user_name = input('Enter desired name:\n')
if user_name in legal_names:
    print(f'{user_name} is an acceptable Danish name. Congratulations.')
else:
    print(f'{user_name} is not acceptable.')
    for name in legal_names:
        if user_name[0] == name[0]:
            print(f'You might consider: {name},', end=' ')
            break
```


Bjork

Run

PARTICIPATION ACTIVITY 23.8.1: Loop else.

```
x = 0
y = 5
z = ?
while x < y:
    if x == z:
        print('x == z')
        break
    x += 1
else:
    print('x == y')
```


1) What is the output of the code if z is 3?

- x == z
- x == y

2) What is the output of the code if z is 10?

- x == z
- x == y

CHALLENGE ACTIVITY 23.8.1: Loop else.

422102.2723990.qx3zqy7

Start

Type the program's output

```
result = 0
n = 2
while n > -2:
    print(n, end=' ')
    result -= 4
    n -= 1
else:
    print(f'/ {result}')
print('done')
```


1 2 3

Check Next

23.9 Getting both index and value when looping: enumerate()

The enumerate() function

A programmer commonly requires both the current position index and corresponding element value when iterating over a sequence. The example below demonstrates how using a for loop with range() and len() to iterate over a sequence generates a position index but requires extra code to retrieve a value.

Figure 23.9.1: Using range() and len() to iterate over a sequence.

```
origins = [4, 8, 10]

for index in range(len(origins)):
    value = origins[index]  # Retrieve value of element
in list.
    print(f'Element {index}: {value}')
```

```
Element 0: 4
Element 1: 8
Element 2:
10
```

Similarly, a for loop that iterates over a container obtains the value directly, but must look up the index with a function call.

Figure 23.9.2: Using list.index() to find the index of each element.


```
origins = [4, 8, 10]

for value in origins:
    index = origins.index(value)  # Retrieve index of
value in list
    print(f'Element {index}: {value}')
```

```
Element 0:
4
Element 1:
8
Element 2:
10
```

The ***enumerate()*** function retrieves both the index and corresponding element value at the same time, providing a cleaner and more readable solution.

Figure 23.9.3: The enumerate() function.

```
origins = [4, 8, 10]

for (index, value) in
enumerate(origins):
    print(f'Element {index}: {value}')
```

```
Element 0: 4
Element 1: 8
Element 2:
10
```


The enumerate() function yields a new tuple each iteration of the loop, with the tuple containing the current index and corresponding element value. In the example above, the for loop *unpacks* the tuple yielded by each iteration of `enumerate(origins)` into two new variables: "index" and "value". ***Unpacking*** is a process that performs multiple assignments at once, binding comma-separated names on the left to the elements of a sequence on the right. Ex: `num1, num2 = [350, 400]` is equivalent to the statements `num1 = 350` and `num2 = 400`.

PARTICIPATION ACTIVITY 23.9.1: enumerate().

Use the following code to answer the question below:

```
for (index, value) in enumerate(my_list):
    print(index, value)
```

1) If `my_list = ['Greek', 'Nordic', 'Mayan']`, what is the output of the program?

- Greek
  Nordic
  Mayan
- 0 Greek
  1 Nordic
  2 Mayan
- 1 Greek
  2 Nordic
  3 Mayan

CHALLENGE ACTIVITY | 23.9.1: Using enumerate in for loops.

422102.2723990.qx3zqy7

Start

Type the program's output


```
colors = ['red', 'green', 'yellow', 'black']
for (position, color) in enumerate(colors):
    print(color, position)
```

1

2

Check Next