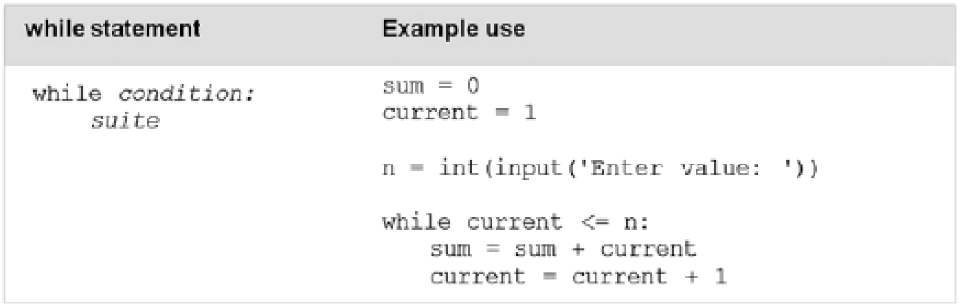
Iterative Control

An **iterative control statement** is a control statement providing the repeated execution of a set of instructions. An *iterative control structure* is a set of instructions and the iterative control statement(s) controlling their execution. Because of their repeated execution, iterative control structures are commonly referred to as “loops.” We look at one specific iterative control statement next, the while statement.

While Statement

A **while statement** is an iterative control statement that repeatedly executes a set of statements based on a provided Boolean expression (condition). All iterative control needed in a program can be achieved by use of the while statement. The table below contains an example of a while loop in Python that sums the first n integers, for a given (positive) value n entered by the user.

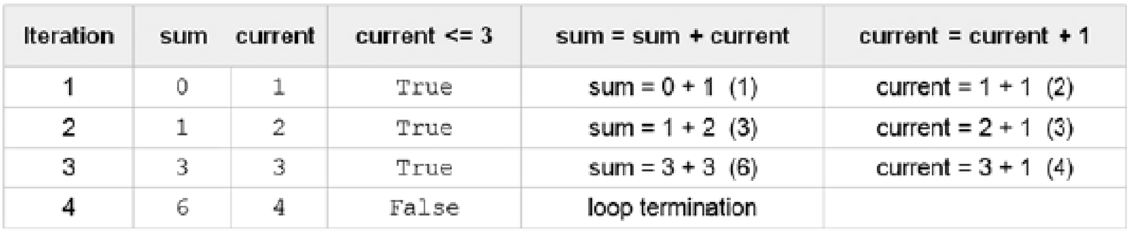


while statement	Example use

`while condition:`
`    suite`
	`sum = 0`
	`current = 1`
	
	`n = int(input('Enter value: '))`
	
	`while current <= n:`
	`    sum = sum + current`
	`    current = current + 1`

As long as the condition of a while statement is true, the statements within the loop are (re)executed. Once the condition becomes false, the iteration terminates and control continues with the first statement after the while loop. Note that it is possible that the first time a loop is reached, the condition may be false, and therefore the loop would never be executed.

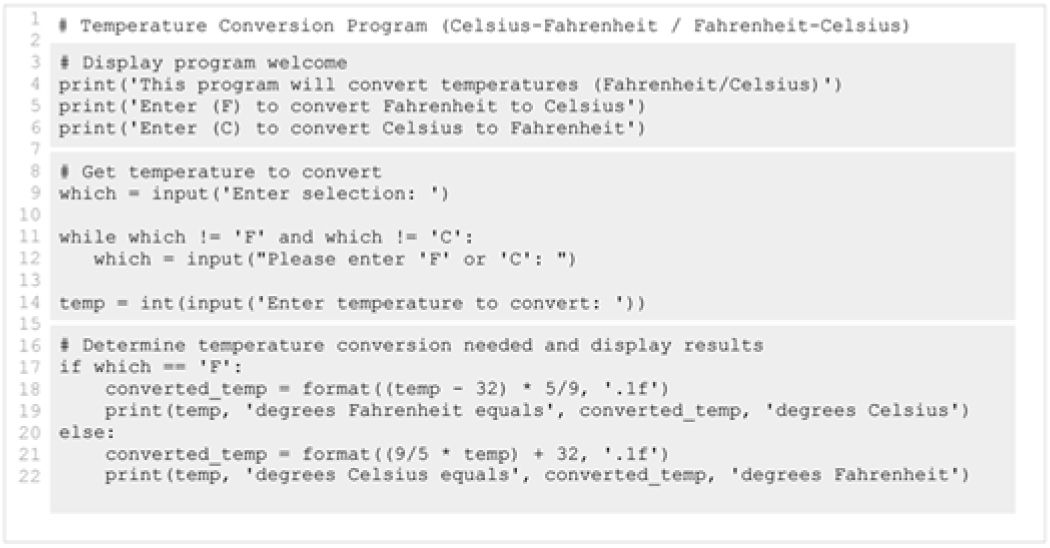
Suppose, for the example in the figure, that the user enters the value 3. Since variable `current` is initialized to `1` (referred to as a *counter variable*), the first time the while statement is reached, `current ,5 3` is true. Thus, the statements within the loop are executed and `sum` is updated to `sum 1 current`. Since `sum` is initialized to `0`, `sum` becomes `1`. Similarly, `current` is updated and assigned to `2`. After the first time through the loop, control returns to the “top” of the loop. The condition is again found to be true and thus the loop is executed a second time. In this iteration, both `sum` and `current` become `3`. In the next iteration, the condition is still true, and therefore, the loop is executed a third time. This time, `sum` becomes `6` and `current` becomes `4`. Thus, when control returns to the top of the loop, the condition is `False` and the loop terminates. The final value of `sum` therefore is `6(1 1 2 1 3)`. This process is summarized below:



Iteration	sum	current	current <= 3	sum = sum + current	current = current + 1
1	0	1	True	sum = 0 + 1 (1)	current = 1 + 1 (2)
2	1	2	True	sum = 1 + 2 (3)	current = 2 + 1 (3)
3	3	3	True	sum = 3 + 3 (6)	current = 3 + 1 (4)
4	6	4	False	loop termination	

Input Error Checking

The while statement is well suited for input error checking in a program. This is demonstrated in the revised version of the temperature conversion program, reproduced below:



```
# Temperature Conversion Program (Celsius-Fahrenheit / Fahrenheit-Celsius)

# Display program welcome
print('This program will convert temperatures (Fahrenheit/Celsius)')
print('Enter (F) to convert Fahrenheit to Celsius')
print('Enter (C) to convert Celsius to Fahrenheit')

# Get temperature to convert
which = input('Enter selection: ')

while which != 'F' and which != 'C':
   which = input("Please enter 'F' or 'C': ")

temp = int(input('Enter temperature to convert: '))

# Determine temperature conversion needed and display results
if which == 'F':
    converted_temp = format((temp - 32) * 5/9, '.1f')
    print(temp, 'degrees Fahrenheit equals', converted_temp, 'degrees Celsius')
else:
    converted_temp = format((9/5 * temp) + 32, '.1f')
    print(temp, 'degrees Celsius equals', converted_temp, 'degrees Fahrenheit')
```

The difference in this program from the previous version is that rather than terminating on invalid input, the program continues to prompt the user until a valid temperature conversion, `'F'` or `'C'`, is entered. Thus, the associated `input` statement is contained within a while loop that keeps iterating as long as variable `which` contains an invalid value. Once the user enters a proper value, the loop terminates allowing the program to continue.

Your Turn

In IDLE, create and run a program containing the code below and observe the results. Make sure to indent exactly.

```
n = 10
sum = 0
current = 1
while current <= n:
     sum = sum + current
     current = current + 1
print (sum)
```

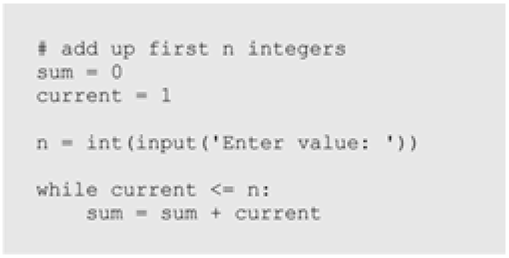
???

```
n = 10
sum = 0
current = 1
while current <= n:
     sum = sum + current
     current = current + 1
     print (sum)
```

???

Part II - Infinite loops

An **infinite loop** is an iterative control structure that never terminates (or eventually terminates with a system error). Infinite loops are generally the result of programming errors. For example, if the condition of a while loop can never be false, an infinite loop will result when executed. Consider if the program segment in Figure 3-17, reproduced in Figure 3-20, omitted the statement incrementing variable `current`. Since `current` is initialized to `1`, it would remain `1` in all iterations, causing the expression `current ,5 n` to be always be true. Thus, the loop would never terminate.



```
# add up first n integers
sum = 0
current = 1

n = int(input('Enter value: '))

while current <= n:
    sum = sum + current
```

Such infinite loops can cause a program to "hang," that is, to be unresponsive to the user. In such cases, the program must be terminated by use of some special keyboard input (such as ctrl-C) to interrupt the execution.

Your Turn

In IDLE, create and run a program containing the code below and observe the results. Make sure to indent the code exactly as shown. To terminate an executing loop, hit ctrl-C.

```
while True:
     print ('Looping')
???
n = 10                          n = 10
sum = 0                         sum = 0
current = 1                     current = 1
while current <= n:             while current <= n:
     sum = sum + current             sum = sum + current
     current = current + 1           current = current + 1
print (sum)                          print (sum)

???                            ???
```

Part III - Definite vs. Indefinite Loops

A **definite loop** is a program loop in which the number of times the loop will iterate can be determined before the loop is executed. For example, following code is a definite loop,

```
sum = 0
current = 1
n = input('Enter value: ')
while current <= n:
    sum = sum + current
    current = current + 1
```

Although it is not known what the value of n will be until the input statement is executed, its value
is known by the time the while loop is reached. Thus, it will execute "n times."

An **indefinite loop** is a program loop in which the number of times that the loop will iterate cannot be determined before the loop is executed. Consider the while loop in the temperature conversion program.

```
which = input("Enter selection: ")
while which != 'F' and which != 'C':
    which = input("Please enter 'F' or 'C': ")
```

In this case, the number of times that the loop will be executed depends on how many times the user mistypes the input. Thus, a while statement can be used to construct both definite and indefinite loops. In the next chapter we look at the for statement, specifically suited for the construction of definite loops.

Boolean Flags and Indefinite Loops

Often the condition of a given while loop is denoted by a single Boolean variable, called a **Boolean flag**. This is shown in the sample code on the next page

Boolean variable valid_entries is a Boolean flag, controlling the while loop at line 12. If the mileage of the last oil change is greater than the current mileage, an error message is displayed (lines 17–18), and the while loop is re-executed. If the current mileage is greater than (or equal to) the mileage of the last oil change, miles_traveled is set to this difference and valid_ entries is set to True, causing the loop to terminate. Thus, lines 23–28 display either that they are due for an oil change, an oil change will soon be needed, or there is no immediate need for an oil change.

```python
# Oil Change Notification Program

# display program welcome
print('This program will determine if your car is in need of an oil change')

# init
miles_between_oil_change = 7500  # num miles between oil changes
miles_warning = 500      # how soon to warn of needed oil change
valid_entries = False

# get mileage of last oil change and current mileage and display
while not valid_entries:
    mileage_last_oilchange = int(input('Enter mileage of last oil change: '))
    current_mileage = int(input('Enter current mileage: '))

    if current_mileage < mileage_last_oilchange:
        print('Invalid entry - current mileage entered is less than')
        print('mileage entered of last oil change')
    else:
        miles_traveled = current_mileage - mileage_last_oilchange
        valid_entries = True

if miles_traveled >= miles_oil_change:
    print('You are due for an oil change')
elif miles_traveled >= miles_oil_change - miles_warning:
    print('You will soon be due for an oil change')
else:
    print('You are not in immediate need of an oil change')
```

Concepts and Procedures

1. A while loop continues to iterate until its condition becomes false. TRUE/FALSE

2. A while loop executes zero or more times. TRUE/FALSE

3. All iteration can be achieved by a while loop. TRUE/FALSE

4. An infinite loop is an iterative control structures that,
   a) Loops forever and must be forced to terminate
   b) Loops until the program terminates with a system error
   c) Both of the above

5. The terms definite loop and indefinite loop are used to indicate whether,
   a) A given loop executes at least once
   b) The number of times that a loop is executed can be determined before the loop is executed.
   c) Both of the above

6. A Boolean flag is,
   a) A variable
   b) Has the value `True` or `False`
   c) Is used as a condition for control statements
   d) All of the above

Problem Solving

1. Write a program segment that uses a `while` loop to add up all the even numbers between 100 and 200, inclusive.

2. The following `while` loop is meant to multiply a series of integers input by the user, until a sentinel value of 0 is entered. Indicate any errors in the code given.

```
product = 1
num = input('Enter first number: ')
while num != 0:
        num = input('Enter first number: ')
        product = product * num
    print('product = ', product)
```

3. For each of the following, indicate which is a definite loop, and which is an indefinite loop.

a)
```
num = input('Enter a non-zero value: ')
while num 55 0:
     num = input('Enter a non-zero value: ')
```

b)
```
num = 0
while n < 10:
    print 2 ** n
    n = n + 1
```