Every Boilermaker Engineer Codes: 101
Entry-Level Programming in Python

Lecture 03

Dr. John H. Cole
<jhcole@purdue.edu>

PURDUE
UNIVERSITY®

COLLEGE OF ENGINEERING

Spring 2021

Part I

Repetition Structures

Control Structures

Control structures determine the order in which a set of statements are executed.

Sequence Structure default, statements that execute in the order they appear

Conditional Structure statements execute only if a condition is met

Repetition Structure statements execute repeatedly

- also known as loop structure, or iteration structure
- statements executed over and over until some condition is met

Repetitive Code Example

Write a program that adds three user-specified numbers together.

Editor - add_three.py

```
1 total = 0
2 total += int(input('Enter number: '))
3 total += int(input('Enter number: '))
4 total += int(input('Enter number: '))
5 print(f'The total is {total}')
```

Terminal

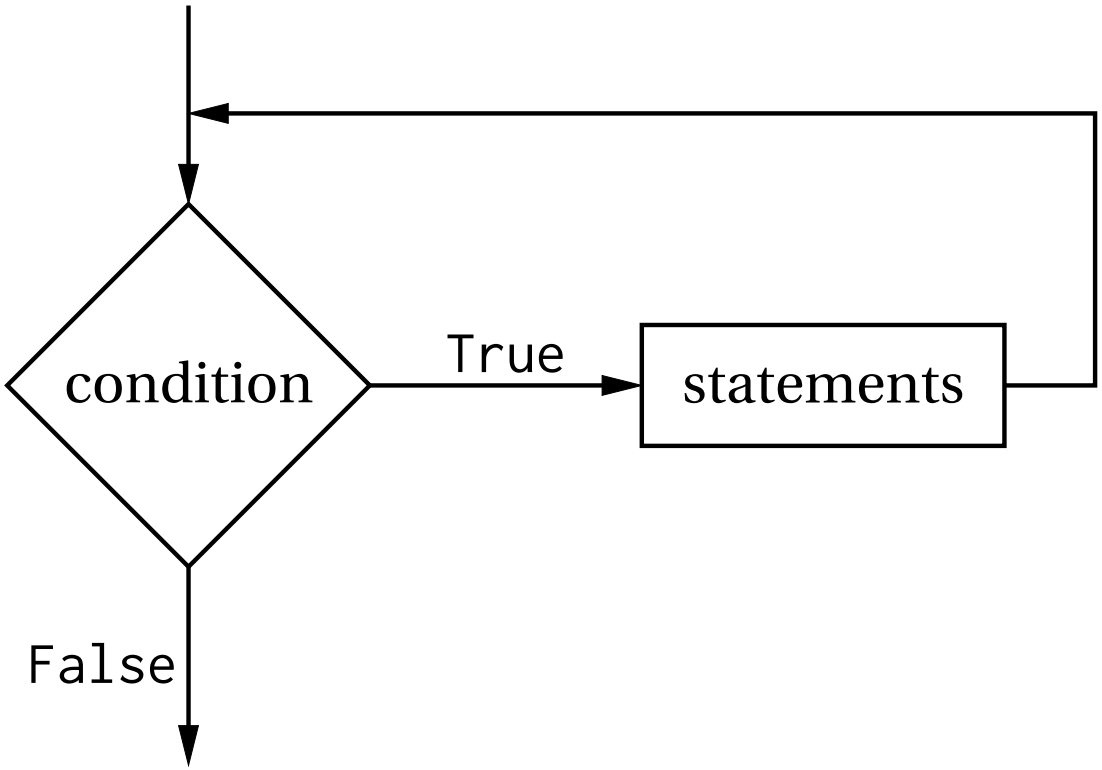
```
$ python add_three.py
Enter number: 10
Enter number: 12
Enter number: 14
The total is 36
```

Time consuming to:

- add more numbers (e.g. 100)
- modify the prompt ('Please')
- fix a bug (change int to float)

Repetition Structure

- implemented in code via the `while` or `for` statements
- each pass through the loop body is called an **iteration**
- a loop **control variable** determines the number of iterations



The `while` Loop

Editor - while_syntax.py

```
1 statement_1
2
3 while condition:
4     statement_2
5     statement_3
6
7 statement_4
```

`statement_1` always executes first

`statement_2` executes repeatedly until condition is `False`

`statement_3` executes repeatedly until condition is `False`

`statement_4` always executes last

- condition evaluates to either `True` or `False`
- the `while` statement ends with a colon ':'
- indented statement block executes while condition is `True`

Repetitive Code Example

Editor - add_three.py

```
1 MAX, count, total = 3, 0, 0
2 while count < MAX:
3     count += 1
4     total += int(input('Enter number: '))
5 print(f'The total is {total}')
```

Terminal

```
$ python add_three.py
Enter number: 10
Enter number: 12
Enter number: 14
The total is 36
```

Looping makes it easy to:

- add more numbers (e.g. 100)
- modify the prompt ('Please')
- fix a bug (change int to float)

Basic Loop Structure

A definite while loop performs three actions

1. initialize a control variable before entering the loop
2. check the control variable and enter the loop body if it is `True`
3. update the control variable within the loop so that the loop checked condition eventually becomes `False`

Editor - add_three.py

```
1 MAX, count, total = 3, 0, 0 # initialize count
2 while count < MAX:          # check count
3     count += 1              # update count
4     total += int(input('Enter number '))
5 print('The total is', total)
```

Infinite Loop

How many times will 'Hello World!' be printed?

Editor - hello_loop.py

```
1 count = 0
2 while count < 4:
3     print('Hello World!')
```

- failed to update the control variable (i.e. count is always less than 4)
- results in an infinite loop!
- interrupt the program by typing ctrl + C

Terminal

```
$ python hello_loop.py
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Infinite Loop Fixed

Editor - hello_fixed.py

```
1 count = 0          # init
2 while count < 4:   # check
3     count += 1     # update!
4     print('Hello World!')
```

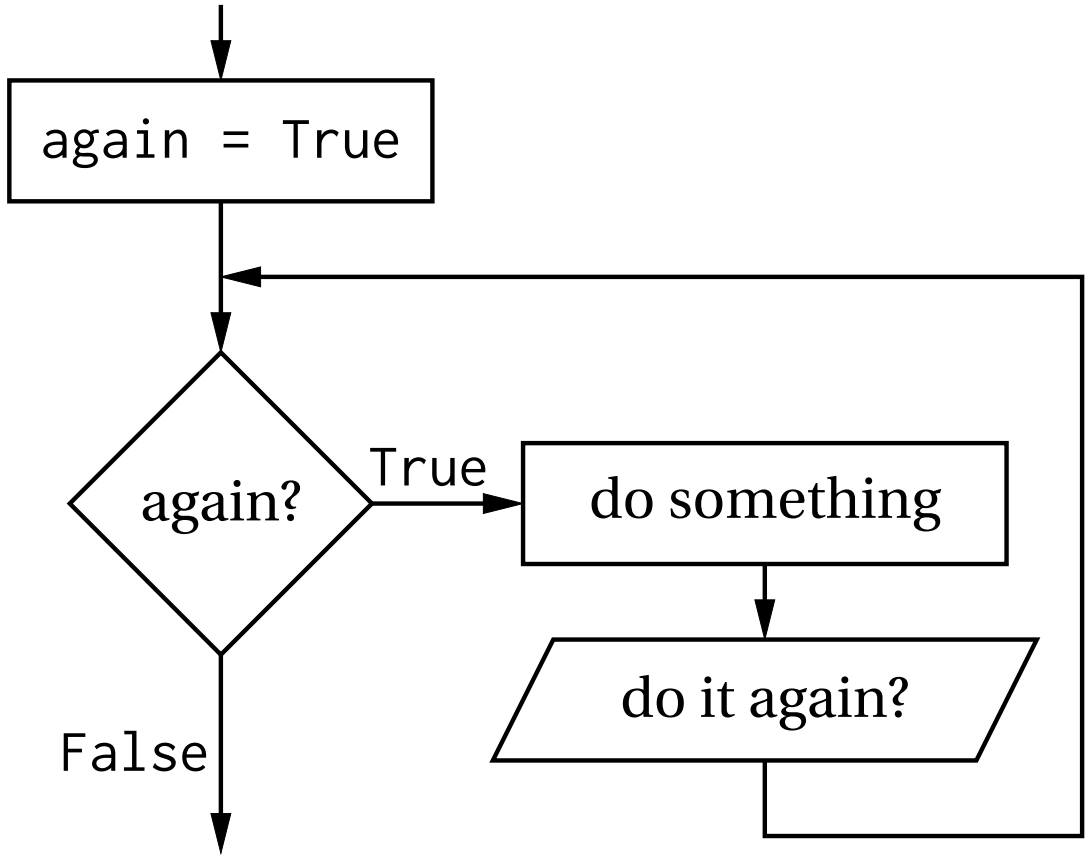
Terminal

```
$ python hello_fixed.py
Hello World!
Hello World!
Hello World!
Hello World!
```

Indefinite Loops

Indefinite Loop A loop in which the number of iterations is not known before the program runs.

- Programmer does not know how many times the loop with execute
- Can use any source of input to set control variable



Indefinite Loop Code Example

- This loop repeats as long as the user enters 'y' at the prompt.

Editor - indef_loop.py

```
1 a = 'y' # initialize
2 while a == 'y': # check
3     print('Pete')
4     # update
5     a = input('repeat? ')
```

Terminal

```
$ python indef_loop.py
Pete
repeat? y
Pete
repeat? y
Pete
repeat? maybe
$
```

Sentinels

Sentinel a distinct value that marks the end of a sequence

- used to exit a loop
- must be distinct from regular sequence values
- examples
  - 0 for a sequence of weights
  - -1 for a sequence of quiz scores
  - `'Q'`, `'q'` or `'quit'` when expecting anything else
  - End-of-file (EOF) when reading from a file

Sentinel - Example

- grades are always positive values
- the sentinel here is -1
- accepts grades until user enters a negative grade

Editor - grade_sentinel.py

```
1 print('Enter -1 to stop')
2 msg = 'Next grade: '
3
4 grade = float(input(msg))
5 while grade >= 0:
6     print(f'Grade is {grade}')
7     grade = float(input(msg))
8
9 print('Done')
```

Terminal

```
Enter -1 to stop
Next grade: 95
Grade is 95.0
Next grade: 100
Grade is 100.0
Next grade: -1
Done
$
```

A Running Total

- often need to calculate a total (e.g. charges on a bill)
- use an accumulator variable inside the loop
- must initialize the accumulator outside the loop

Editor - grade_total.py

```
1 print('Enter -1 to stop')
2 msg = 'Next grade: '
3 total = 0
4 grade = float(input(msg))
5 while grade >= 0:
6     total += grade
7     grade = float(input(msg))
8
9 print(f'Total = {total}')
```

Terminal

```
Enter -1 to stop
Next grade: 95
Next grade: 100
Next grade: -1
Total = 195.0
```

- the sentinel is *not* added to the total

Calculating an Average

- often need to calculate an average (e.g. batting average)
- use an counter variable inside the loop
- must initialize the counter outside the loop

Editor - grade_average.py

```
1 print('Enter -1 to stop')
2 msg = 'Next grade: '
3 count, total = 0, 0
4 grade = float(input(msg))
5 while grade >= 0:
6     count += 1
7     total += grade
8     grade = float(input(msg))
9 print(f'Avg = {total/count}')
```

Terminal

```
Enter -1 to stop
Next grade: 95
Next grade: 100
Next grade: -1
Avg = 97.5
```

Calculating an Average - Division by Zero

- watch out for division by zero

Terminal

```
$ python grade_average.py
Enter -1 to stop
Next grade: -1
Traceback (most recent call last):
  File "grade_average.py", line 9, in <module>
    print(f'Avg = {total/count}')
ZeroDivisionError: division by zero
```

Calculating an Average - Division by Zero Fixed

Editor - grade_average.py

```
1  print('Enter -1 to stop')
2  msg = 'Next grade: '
3  count, total = 0, 0
4  grade = float(input(msg))
5  while grade >= 0:
6      count += 1
7      total += grade
8      grade = float(input(msg))
9  if count > 0:
10     print(f'Avg = {total/count}')
11 else:
12     print('No valid grades')
```

Terminal

```
Enter -1 to stop
Next grade: -1
No valid grades
```

The for Loop

Editor - for_syntax.py

```
1 statement_1
2
3 for item in iterable:
4     statement_2
5     statement_3
6
7 statement_4
```

`statement_1` always executes first

`statement_2` executes once for each item in the iterable

`statement_3` executes once for each item in the iterable

`statement_4` always executes last

- has a fixed number of iterations (the size the iterable)
- an iterable is any object that can be a iterated over (e.g. `string`, `list`, `range`, etc.)
- `item` is set to a different value in the iterable with each iteration

For Loop Examples

Editor - for_word.py

```
1 for c in 'SPAM':
2     print(c)
```

Terminal

```
$ python for_word.py
S
P
A
M
$
```

Editor - for_list.py

```
1 words = ['to', 'be', 'or']
2 for word in words:
3     print(word)
```

Terminal

```
$ python for_list.py
to
be
or
$
```

List Essentials

- surrounded by square brackets [ ]
- items separated by commas
- list items are ordered
- elements can by accessed by index starting at 0
- anything can be in a list, even another list

Terminal

```
>>> l = [3, 5.0, 'spam', None, True, ['a', 'b']]
>>> l[0]
3
>>> l[5]
['a', 'b']
>>>
```

The `range()` Function

- creates a **range object** that a for loop can iterate over
- each iteration, the **range object** produces the next integer in a sequence
- can covert a **range object** to a list with `list()` function
- `range()` function has three forms
  1. `range(stop)`
  2. `range(start, stop)`
  3. `range(start, stop, step)`
- `start`, `stop`, and `step` must be integers
- `stop` value is always excluded from the sequence
- `start` defaults to 0
- `step` defaults to 1

range() Examples

Terminal

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(0,5))
[0, 1, 2, 3, 4]
>>> list(range(0,5,1))
[0, 1, 2, 3, 4]
>>> list(range(0,10,2))
[0, 2, 4, 6, 8]
>>> list(range(-2,5))
[-2, -1, 0, 1, 2, 3, 4]
>>> list(range(8,2,-1))
[8, 7, 6, 5, 4, 3]
```

Terminal

```
>>> list(range(8,2))
[]
>>> list(range(0))
[]
>>> list(range(-2))
[]
>>> list(range(2,2))
[]
>>> list(range(2,2,1))
[]
>>> list(range(1,10,-1))
[]
```

for range() Examples

Editor - squares.py

```
1 # Print a table of squares
2 print(' n  n^2')
3 for n in range(2,11):
4     print(f'{n:2d}  {n**2:3d}')
```

Editor - for_range.py

```
1 # Sum all the even numbers
2 # from 1 to 100
3 total = 0
4 for n in range(0,101,2):
5     total += n
6 print(f'total is {total}')
```

Terminal

```
 n  n^2
 2    4
 3    9
 4   16
 5   25
 6   36
 7   49
 8   64
 9   81
10  100
```

Terminal

```
total is 2550
```

User Variables for Loop Example

Editor - user_squares.py

```
1 # Get limits from the user
2 start = int(input('start: '))
3 stop  = int(input('stop: '))
4
5 # Print a table of squares
6 print(' n   n^2')
7 for n in range(start,stop+1):
8     print(f'{n:2d}  {n**2:4d}')
```

Terminal

```
start: 5
stop: 14
 n   n^2
 5    25
 6    36
 7    49
 8    64
 9    81
10   100
11   121
12   144
13   169
14   196
```

While Loop vs For Loop

Editor - while_loop.py

```
1 n = 0
2 while n < 4:
3     print(n)
4     n += 1
```

Terminal

```
$ python while_loop.py
0
1
2
3
$
```

Editor - for_loop.py

```
1 # initializes, checks,
2 # & updates in one line
3 for n in range(4):
4     print(n)
```

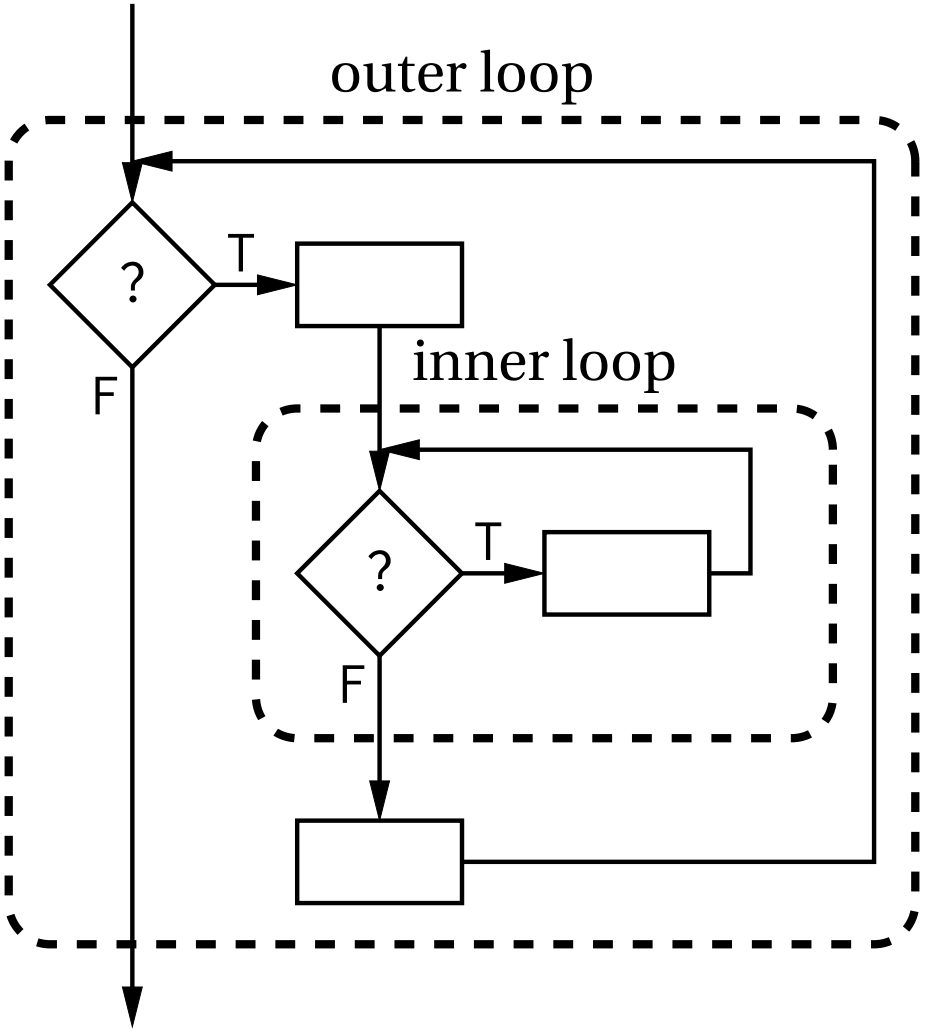
Terminal

```
$ python for_loop.py
0
1
2
3
$
```

Nested Loops

Nested Loop a loop contained in another loop

- inner loop completes all of its iterations for each outer iteration
- total number of inner loop iterations is the product of the iterations of each loop



Nested Loops Clock Example

- hour iterates once for every 60 minutes
- minute iterates once for every 60 seconds
- second iterates once per second

Editor - clock.py

```
from time import sleep
for h in range(12):
    for m in range(60):
        for s in range(60):
            print(f'\r{h:02d}:{m:02d}:{s:02d}', end='')
            sleep(0.001)
print('')
```

Nested Loops Star Grid Example

Editor - star_grid.py

```
1 rows = int(input('rows: '))
2 cols = int(input('columns: '))
3
4 for r in range(rows):
5     for c in range(cols):
6         print('*', end='')
7     print('') # start a new line
```

Terminal

```
rows: 5
columns: 4
****
****
****
****
****
```

r = 4, c = 3

Nested Loops Star Triangle Example

Editor - star_triangle.py

```
1 BASE_SIZE = 5
2 for r in range(BASE_SIZE):
3     for c in range(r + 1):
4         print('*', end='')
5     print('') # start a new line
```

Terminal

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

r = 4, c = 4

Stopping Early: Continue

continue skips to the next iteration of the loop

Editor - continue.py

```
1 for n in range(5):
2     if n == 2:
3         continue
4     print(n)
```

Terminal

```
0
1
3
4
```

Stopping Early: Break

BREAK skips all the remaining iterations in the loop

Editor - break.py

```
1 for n in range(5):
2     if n == 2:
3         break
4     print(n)
```

Terminal

```
0
1
```

Stopping Early: While Break

- continue and break work in while loops too

Editor - while_break.py

```
1 n = 0
2 while n < 5:
3     if n == 2:
4         break
5     print(n)
6     n += 1
```

Terminal

```
0
1
```

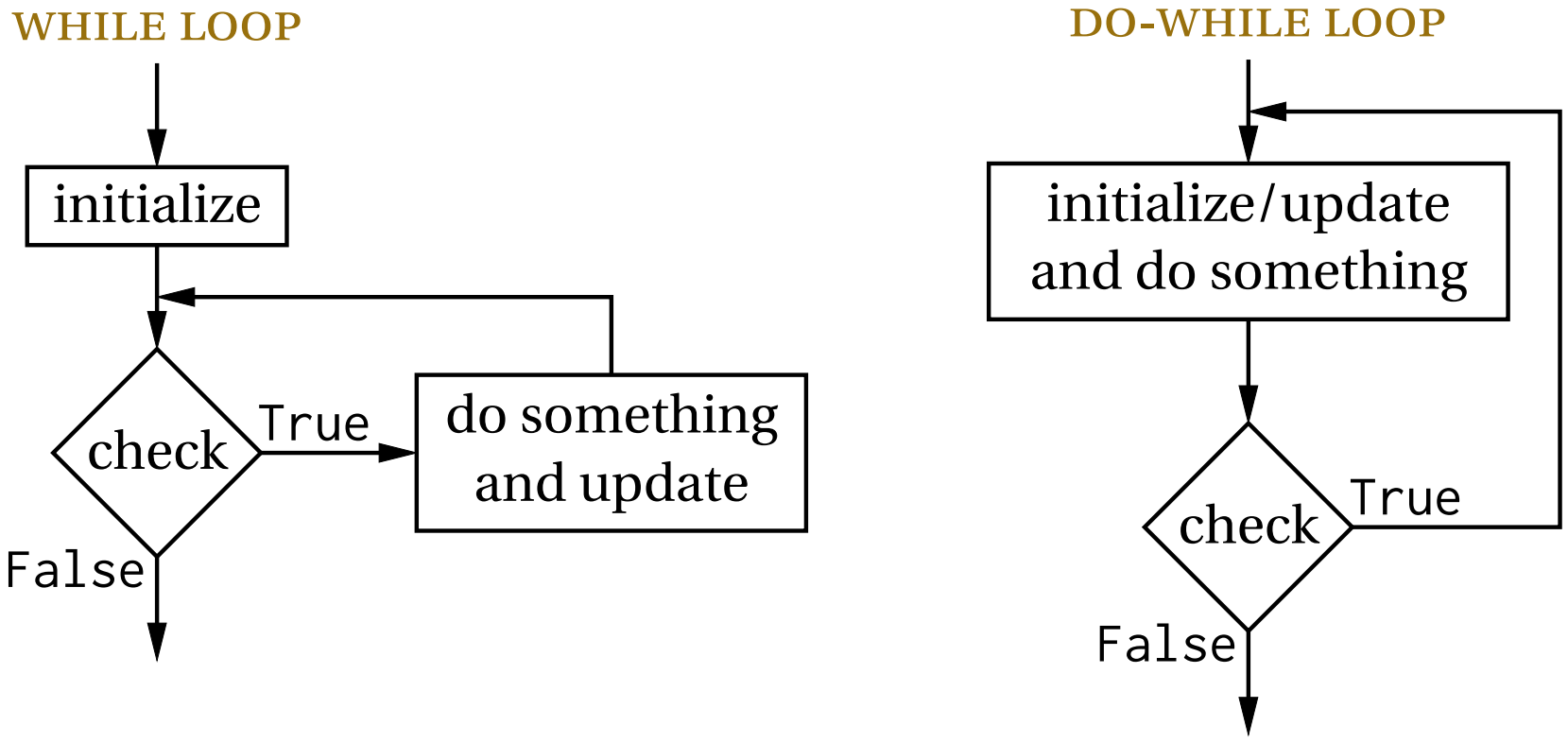
Input Validation While Loop

- can use an indefinite while loop to validate input.
- initialize, check, and update
- the initialize and update steps are often the same

Editor - input_validation_1.py

```
1 value = float(input('Pick a number (1 to 10): '))
2 while value < 1 or 10 < value:
3     print('Invalid Input. Please try again.')
4     value = float(input('Pick a number (1 to 10): '))
```

While vs. Do-While



- Python does not have a built-in `do-while` statement
- implement a do-while loop using `while True:` and `if condition: break`

Input Validation Do-While Loop

Editor - input_validation_2.py

```
1 while True:
2     value = float(input('Pick a number (1 to 10): '))
3     if 1 <= value and value <= 10:
4         break
5     print('Invalid Input. Please try again.')
```

- reduces code repetition (DRY - "don't repeat yourself")
- DRY is better than WET ("write everything twice", "we enjoy typing" or "waste everyone's time")

Part II

Your Turn

Practice Exercise

Write a program that draws the following pattern using for loops (I've named mine `star_pattern.py`).

Terminal

```
$ python star_pattern.py
*******
******
*****
****
***
**
*
```

Practice Exercise Solutions

Editor - star_pattern.py

```
for i in range(7):
    for j in range(i,7):
        print('*',end='')
    print('')
```

Editor - star_pattern.py

```
for i in range(7,0,-1):
    for j in range(i):
        print('*',end='')
    print('')
```

Editor - star_pattern.py

```
for i in range(7):
    msg = ''
    for j in range(i,7):
        msg = msg + '*'
    print(msg)
```

Editor - star_pattern.py

```
for i in range(7):
    print('*'*(7-i))
```


Thanks for watching!