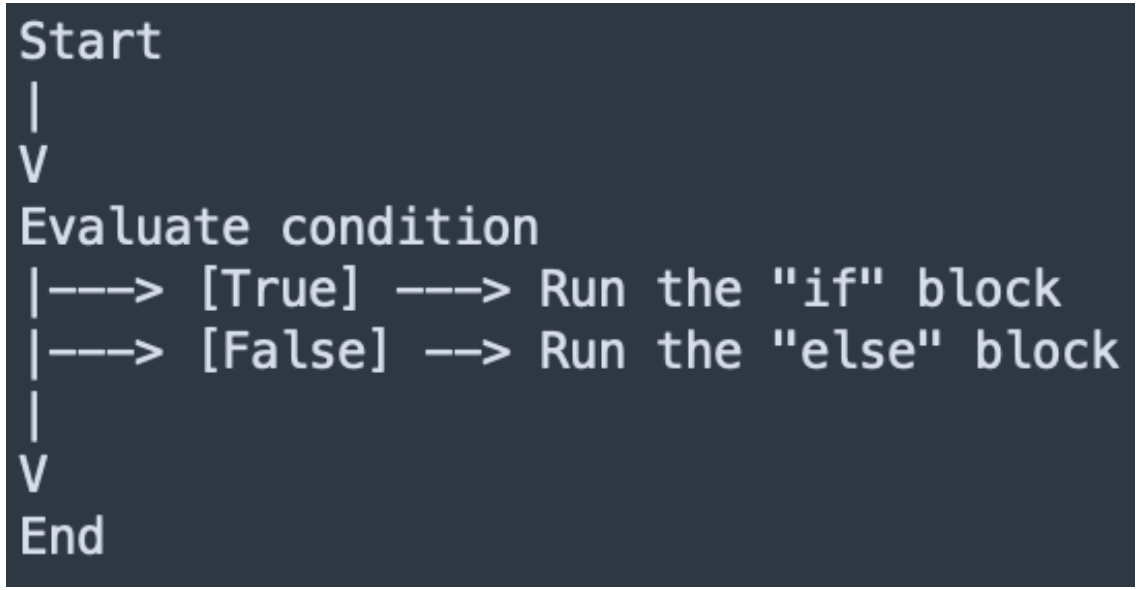
Python: Control Flow

1405 Instructor: Ruiqing (Sam) Cao

Conditionals: `if(-else)` Statements

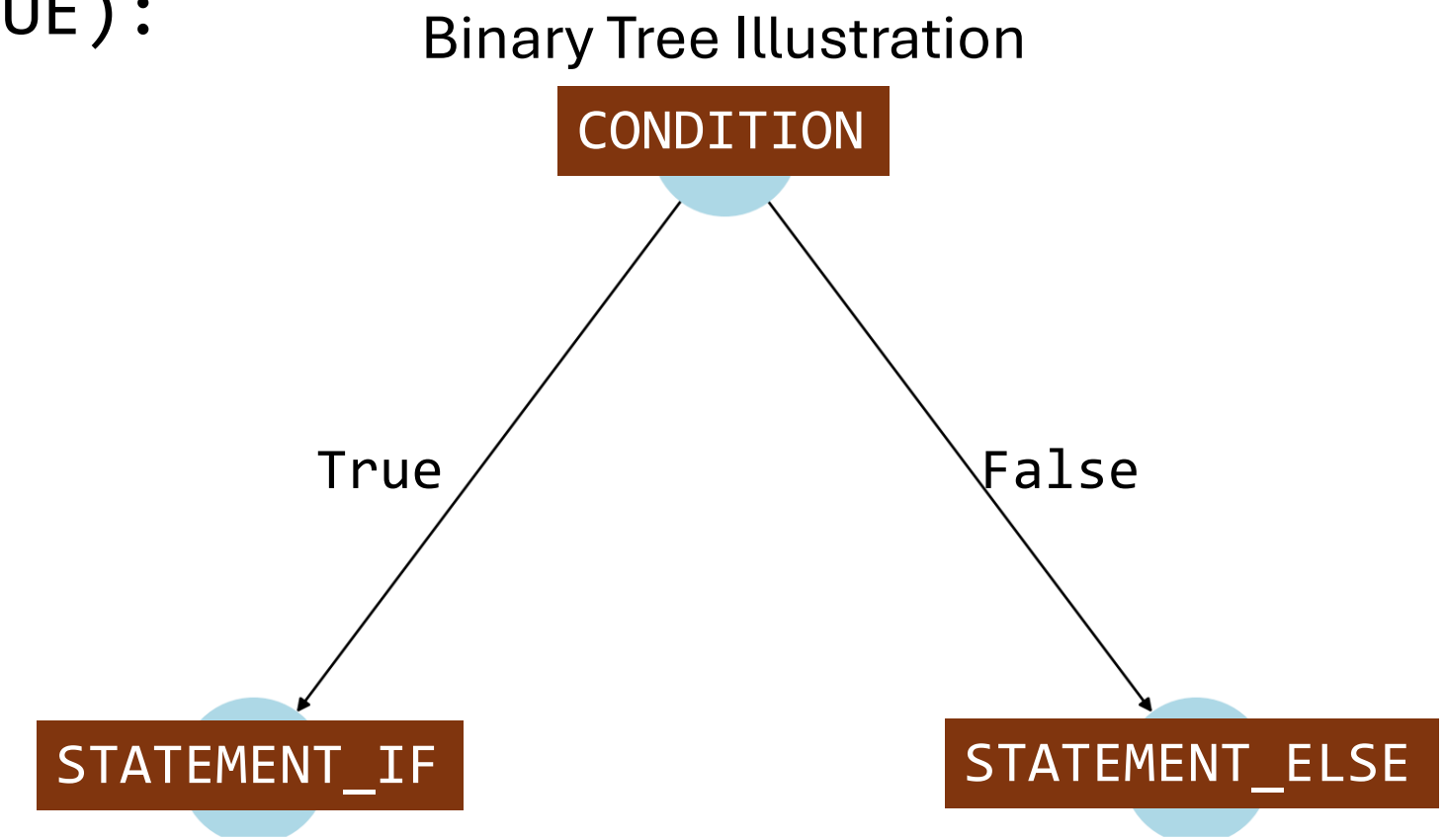
```
if (CONDITION_IS_TRUE):
    STATEMENT_IF
else:
    STATEMENT_ELSE
```

←The **`else`** statement is optional



Conditionals: `if(-else)` Statements

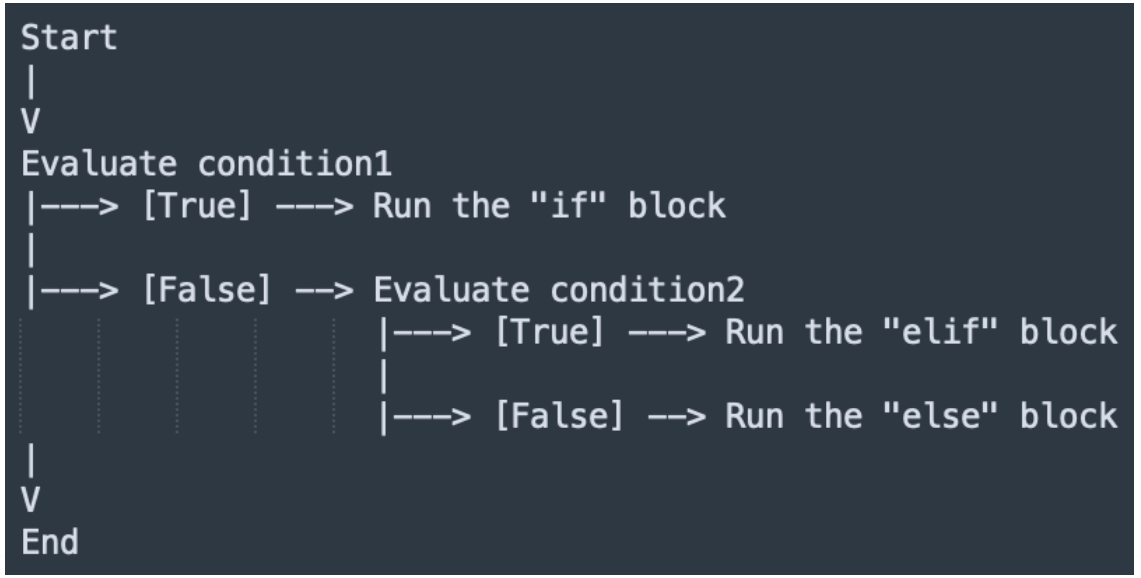
```
if (CONDITION_IS_TRUE):
    STATEMENT_IF
else:
    STATEMENT_ELSE
```



if-elif(-else) Statements

```
if (CONDITION1_IS_TRUE):
    STATEMENT_IF
elif (CONDITION2_IS_TRUE):
    STATEMENT_ELIF
else:
    STATEMENT_ELSE
```

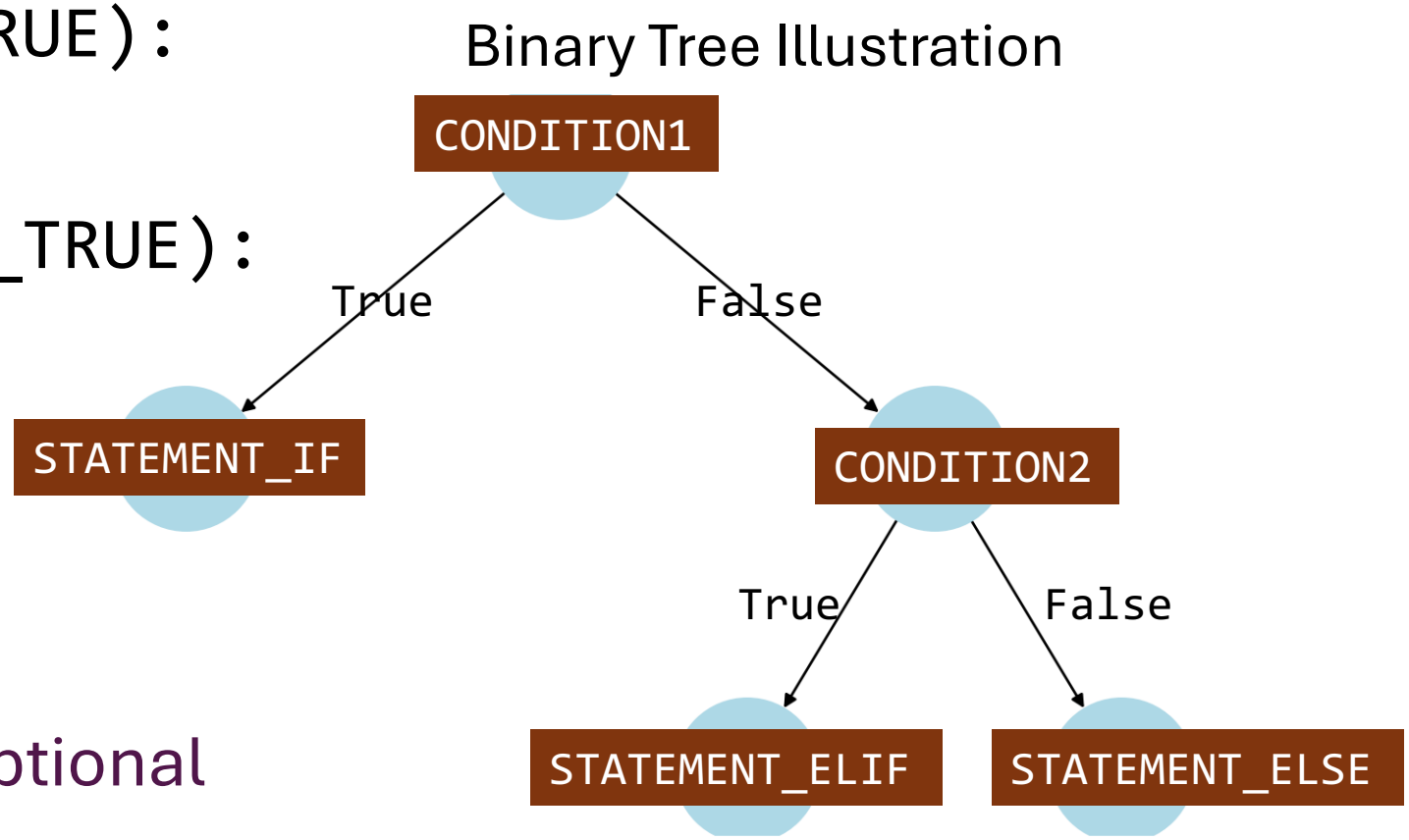
The **else** statement is optional



if-elif(-else) Statements

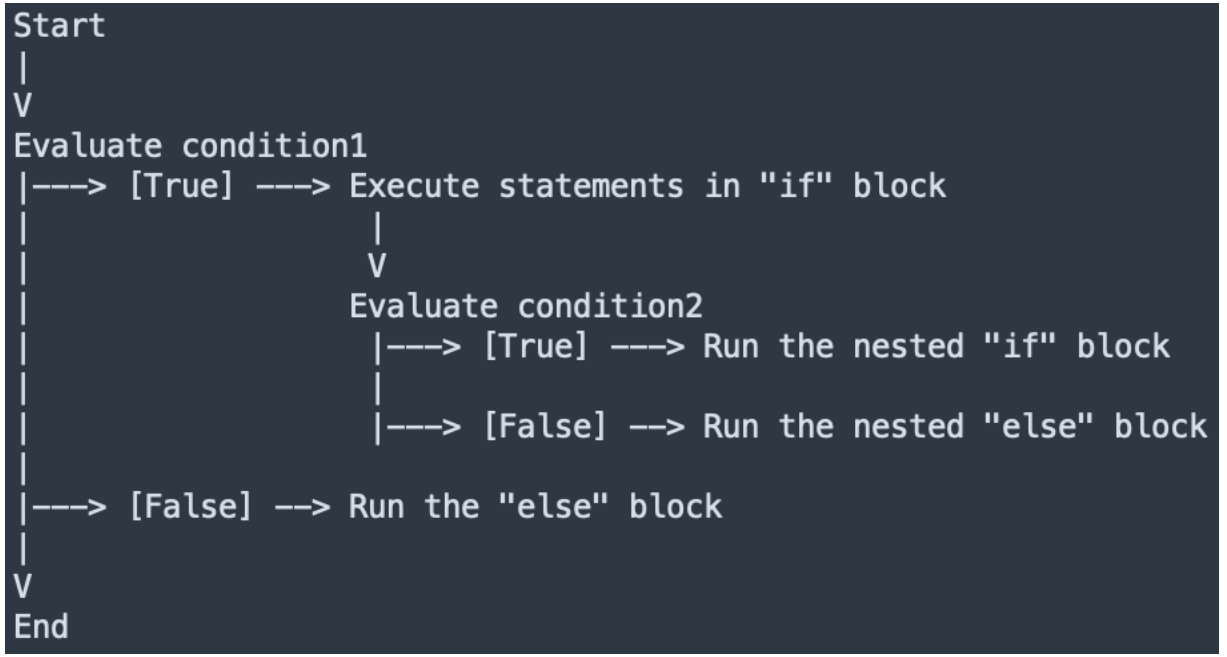
```
if (CONDITION1_IS_TRUE):
    STATEMENT_IF
elif (CONDITION2_IS_TRUE):
    STATEMENT_ELIF
else:
    STATEMENT_ELSE
```

The **else** statement is optional



Nested `if-then(-else)` Statements

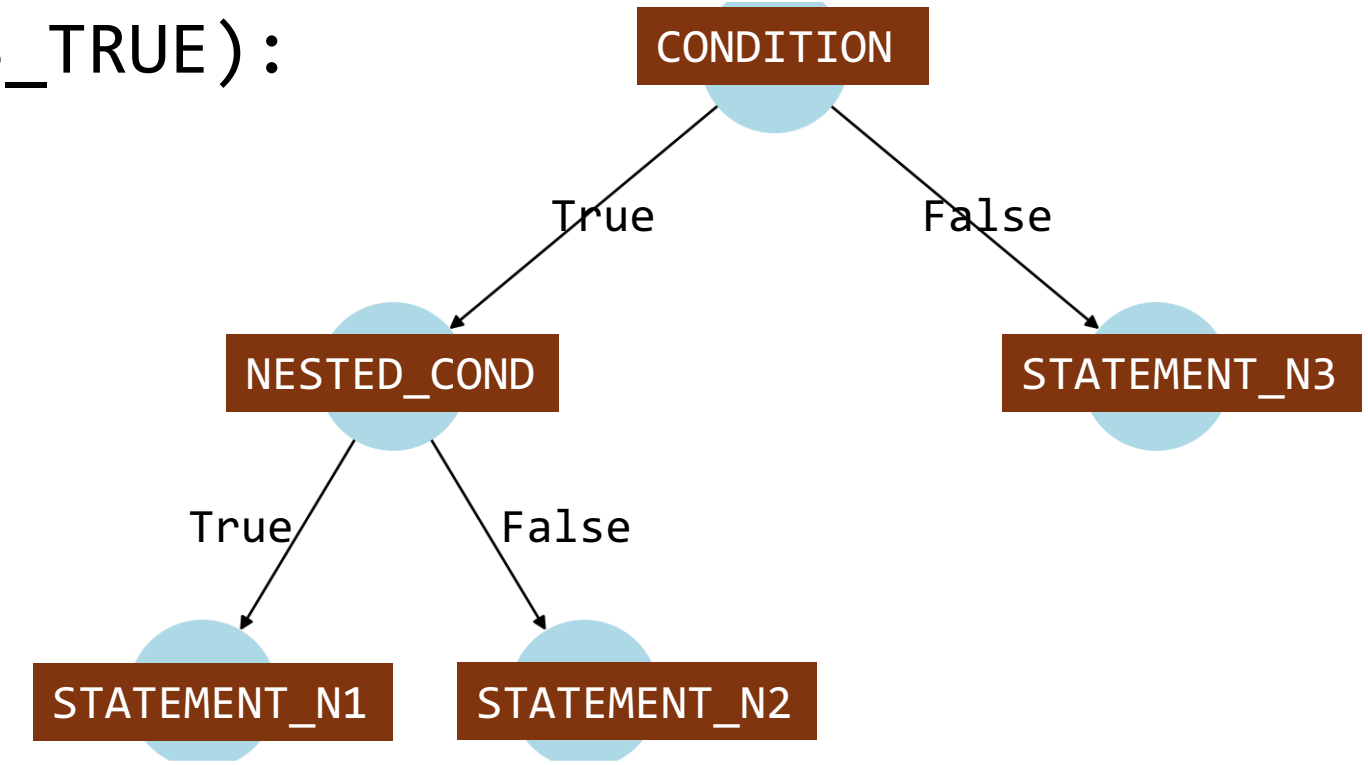
```
if (CONDITION_IS_TRUE):
    if (NESTED_COND_IS_TRUE):
        STATEMENT_N1
    else:
        STATEMENT_N2
else:
    STATEMENT_N3
```



Nested if-then(-else) Statements

```
if (CONDITION_IS_TRUE):
    if (NESTED_COND_IS_TRUE):
        STATEMENT_N1
    else:
        STATEMENT_N2
else:
    STATEMENT_N3
```

Binary Tree Illustration



Indentation is very important

- Indentation provides a visual structure that reflects the semantic structure of the program, and each indented set of expressions denotes a block of instructions

- For instance, the code blocks below have completely different meanings: *the* **`else`** *clause is aligned with different* **`if`** *clauses*

```
if (CONDITION_IS_TRUE):
    if (NESTED_COND_IS_TRUE):
        STATEMENT_N1
    else:
        STATEMENT_ELSE
```

```
if (CONDITION_IS_TRUE):
    if (NESTED_COND_IS_TRUE):
        STATEMENT_N1
else:
    STATEMENT_ELSE
```

Exception Handling

- Corner cases that stop the entire program. You want to avoid this by anticipating all possible conditions before running into them. But sometimes that can make your code unnecessarily verbose
- For example, you wrote a program that inputs two numbers and outputs their sum. But what if something goes wrong, e.g., inputs are not numbers, or the program didn't receive an input?

```
n1 = input("first number: ")
n2 = input("second number: ")
try:
    print(float(n1)+float(n2))
except:
    print("Error")
```

Basic syntax of exception handling to ensure program doesn't stop when it runs into an unexpected error:

```
try:
    ... # main program
except:
    ... # print error message
```

Exercise: Input & Conditionals

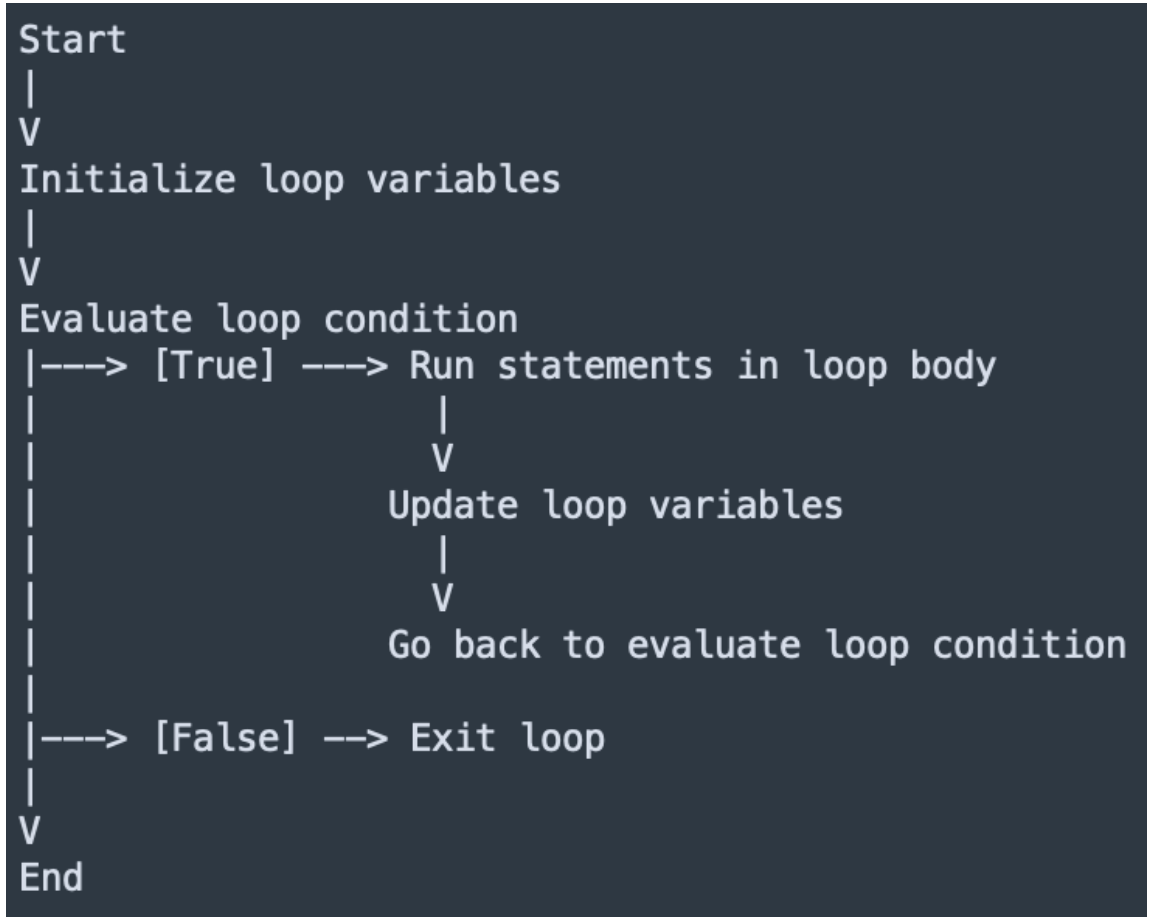
Rain or No Rain? Write a Python program to perform these tasks:

- **Ask the user** to input whether the weather forecast predicts rain today
  - *Hint:* Use a bool type variable to store the user's input
  - Acceptable inputs should include answers like "yes", "no", or similar
- **Print a recommendation** based on the user's input about whether to bring an umbrella when going out
  - *Hint:* Use a string type to generate the recommendation message
- **Handle cases** where the user does not provide relevant information about today's weather
  - In such cases, the program should give a default message advising caution or requesting clearer input

Loops: `for` & `while`

- **Loops** execute a code block repeatedly, for a specific number of times (**`for`** loop) or while a condition is true (**`while`** loop)

- The **`for`** loop iterates over elements of a sequence, such as a `list` or a `string`
- The **`while`** loop repeats itself until a condition is no longer true



for Loop

```
for variable in sequence:
    ...
```

Iterate over a list:

```
firms = ['JPM','Visa','KKR']
for firm in firms:
     print(firm)
```

Iterate over a range:

```
for i in range(10):
     print(i)
```

while Loop

```
while condition:
    ...
```

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

```
password = ""
while password != "1405":
    password = input("Code:")
    print("Access granted!")
```

💡 What is the output from running each of the code blocks above ?

for Loop

```
for variable in sequence:
    ...
```

Iterate over a list:

```
firms = ['JPM','Visa','KKR']
for firm in firms:
     print(firm)
```

Iterate over a range:

```
for i in range(10):
     print(i)
```

0 to 9

while Loop

```
while condition:
    ...
```

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

0 to 9

```
password = ""
while password != "1405":
    password = input("Code:")
    print("Access granted!")
```

💡 What is the output from running each of the code blocks above ?

Loop Control: `break` & `continue`

- The `break` statement halts and exits the current loop (but not the program or outer loops if any): useful for debugging – examine intermediate outputs in the middle of running the program

- The `continue` statement skips the rest of the current iteration and moves directly to the next iteration

- Often combined with `if` statement: exit the current loop (`break`) or skip current iteration (`continue`) when some condition is met

`break` statement

```
i=0
while i<10:
    if i==3:
        break
    print(i)
    i += 1
```

```
for i in range(10):
    if i==3:
        break
    print(i)
```

`continue` statement

```
i=0
while i<5:
    i += 1
    if i==3:
        continue
    print(i)
```

```
for i in range(5):
    if i==3:
        continue
    print(i)
```

💡 What is the output from running each of the code blocks above ?

`break` statement

```
i=0
while i<10:
    if i==3:
        break
    print(i)
    i += 1
```

```
0
1
2
```

`continue` statement

```
i=0
while i<5:
    i += 1
    if i==3:
        continue
    print(i)
```

```
1
2
4
5
```

```
for i in range(10):
    if i==3:
        break
    print(i)
```

```
0
1
2
```

```
for i in range(5):
    if i==3:
        continue
    print(i)
```

```
0
1
2
4
```

💡 What is the output from running each of the code blocks above ?

Exercise: Loop Control

1.
```
s = 0
for i in range(10):
    if (i % 2 == 1):
        s += i**2
    elif i >= 5:
        break
print(s)
```

- What does the code above do?
- What is the final output?

2.
```
j = 0
while j < 10:
    j += 1
    if j >= 5:
        continue
    print(j)
print(j)
```

- What does the code above do?
- What is the final output?

3. Write a program to calculate and print the total sum of all integers from 0 to 20. Complete this task using (1) a `for` loop (2) a `while` loop