ShiP.py

Learn to Py while Shelter-in-Place

L2: Boolean Decisions (Branching)

A volunteering educational initiative during COVID-19

ShiP Crew

JD

Teddy

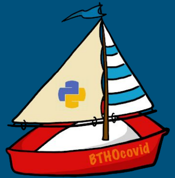
Chinmay

Pratik

Siddharth

Umang

Waseem



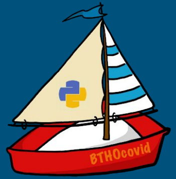
A volunteering educational initiative during COVID-19

Topics

PHASE I: Foundations

All times are in CDT (GMT-5)

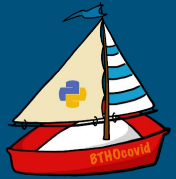
1.	Variables, Expressions, Simple I/O	Sat, April 18 (11 am-12 noon)	☀
2.	Boolean Decisions (branching)	Wed, April 22 (9 pm-10 pm)	🌙
3.	Repetitions (loops)	Sat, April 25 (11 am-12 noon)	☀
4.	Collective Data Structures	Wed, April 29 (9 pm-10 pm)	🌙
5.	Functions	Sat, May 02 (11 am-12 noon)	☀
6.	File I/O	Wed, May 06 (9 pm-10 pm)	🌙
7.	X	Sat, May 09 (11 am-12 noon)	☀



Lecture 2

AGENDA

- Boolean Logic & Truth Tables
- Boolean Operators
- Writing Boolean Expressions
- Boolean Decisions
- Conditional Assignment



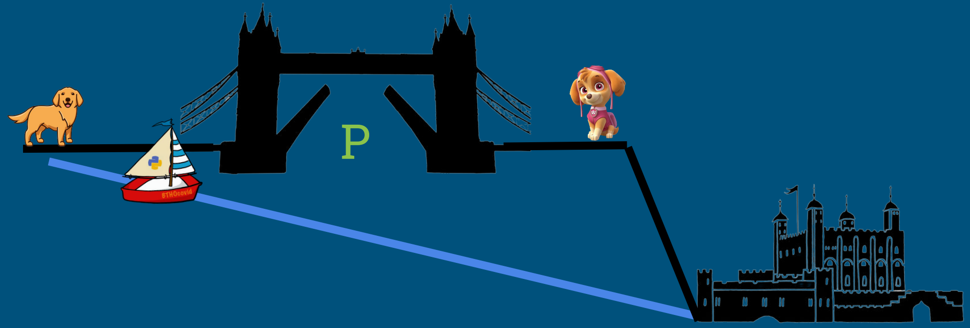
Boolean Logic & Truth Tables

- At the heart of Boolean Logic is that all values (in our case **bool** types variables) are either **True** or **False**.

- There is no other value.

- You can combine one boolean value/expression with another using three fundamental operators **not**, **and**, **or**

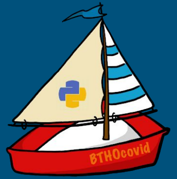
- Certain rules for combining two values/expressions **P**, **Q** (Truth Tables)



P: Bridge bascules DOWN

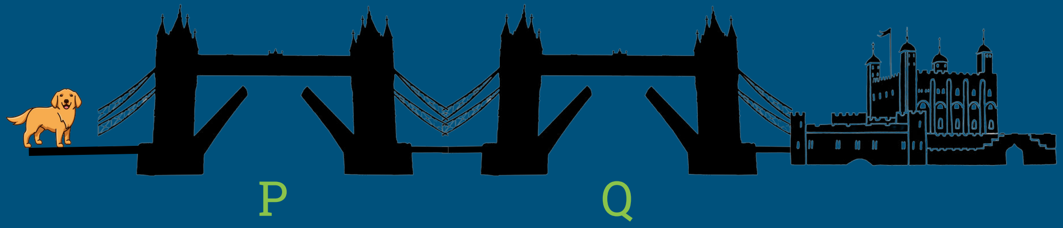
P	
False	True
True	False

not P



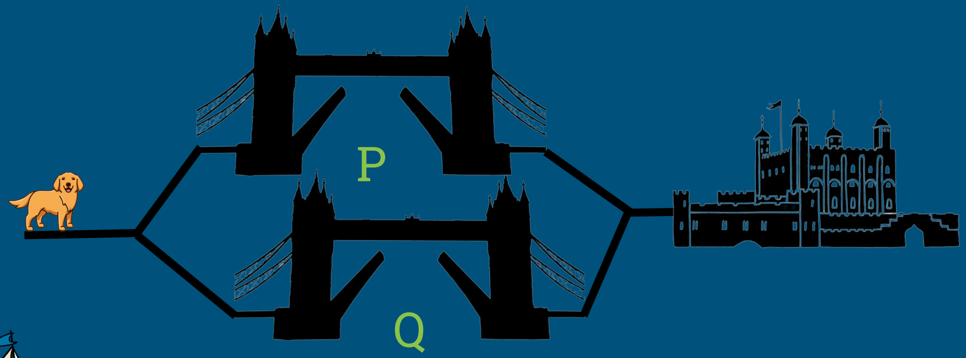
P: Bridge P bascules DOWN

Q: Bridge Q bascules DOWN



P	Q	🏰
False	False	False
False	True	False
True	False	False
True	True	True

P and Q

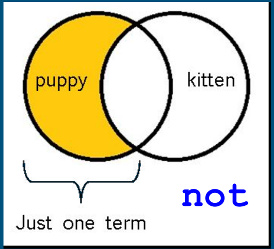


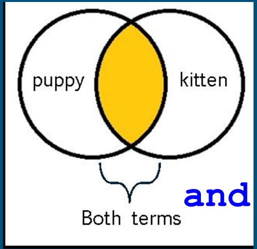
P	Q	🏰
False	False	False
False	True	True
True	False	True
True	True	True

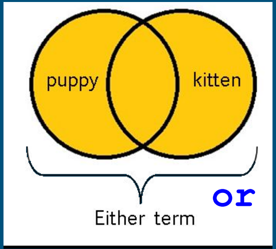
P or Q

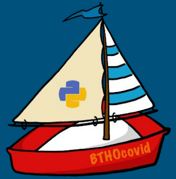
Boolean Operators

Operator	Example	Result/Meaning
`not`	`not P`	*True* if P is *False*. *False* if P is *True* (Inversion)
`and`	`P and Q`	*True* if both P and Q are *True*. *False* otherwise
`or`	`P or Q`	*True* if either P or Q is *True*. *False* if both are *False*







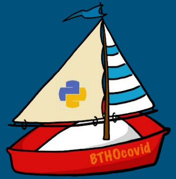


Boolean Operator Precedence

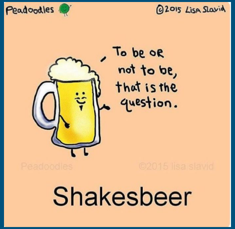
Highest

Lowest

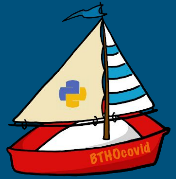
Operator	Description
`( )`	Parenthesis
`**`	Exponentiation
`*, //, /, %`	Multiplication, Division, Modulo
`+, -`	Addition, Subtraction
`==, !=, >, <, >=, <=`	Relational Operators
not	**Boolean NOT**
and	**Boolean AND**
or	**Boolean OR**
`=`	Assignment



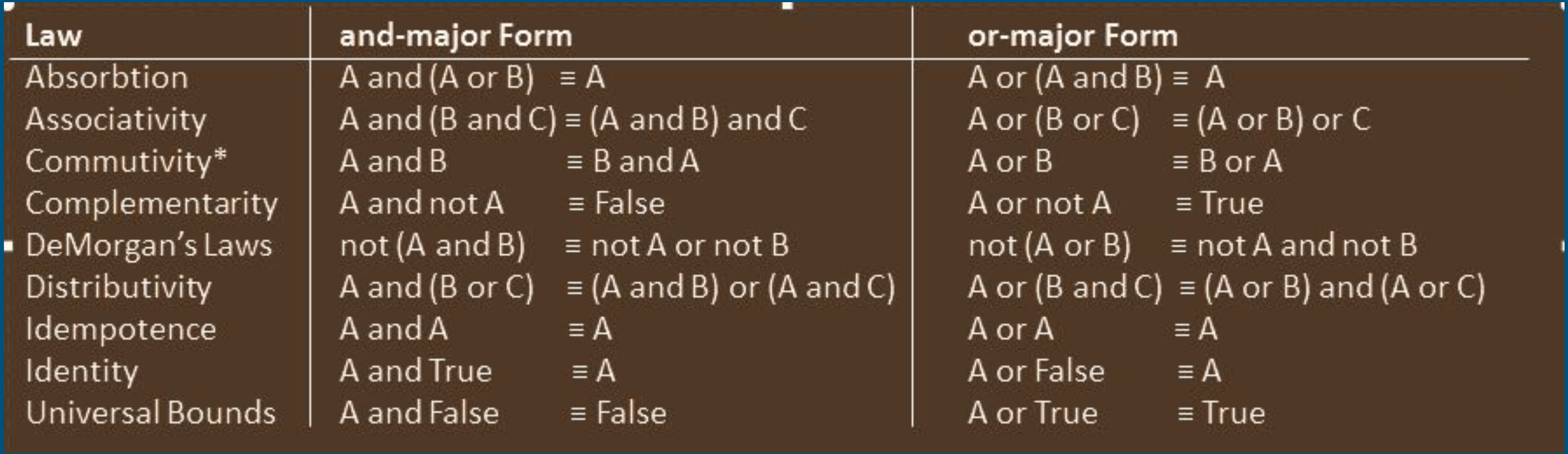
Writing Boolean Expressions



Phrase	Boolean variables	Boolean Expression
The door is not locked	door_locked	not door_locked
If the sky is clear and it is daytime, the sun is shining	sky_clear, daytime, sun_shining	sun_shining = sky_clear and daytime
I can float if I am in a boat or I can swim	in_boat, can_swim, floating	floating = in_boat or can_swim
Order a pizza if there is no food at home and you have friends coming over or if you have a date and favorite Netflix movie planned	food, friends, date, netflix, pizza	pizza = (not food and friends) or (date and netflix)

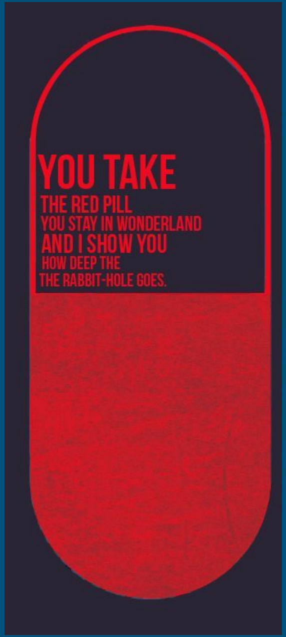


Boolean Algebraic Identities

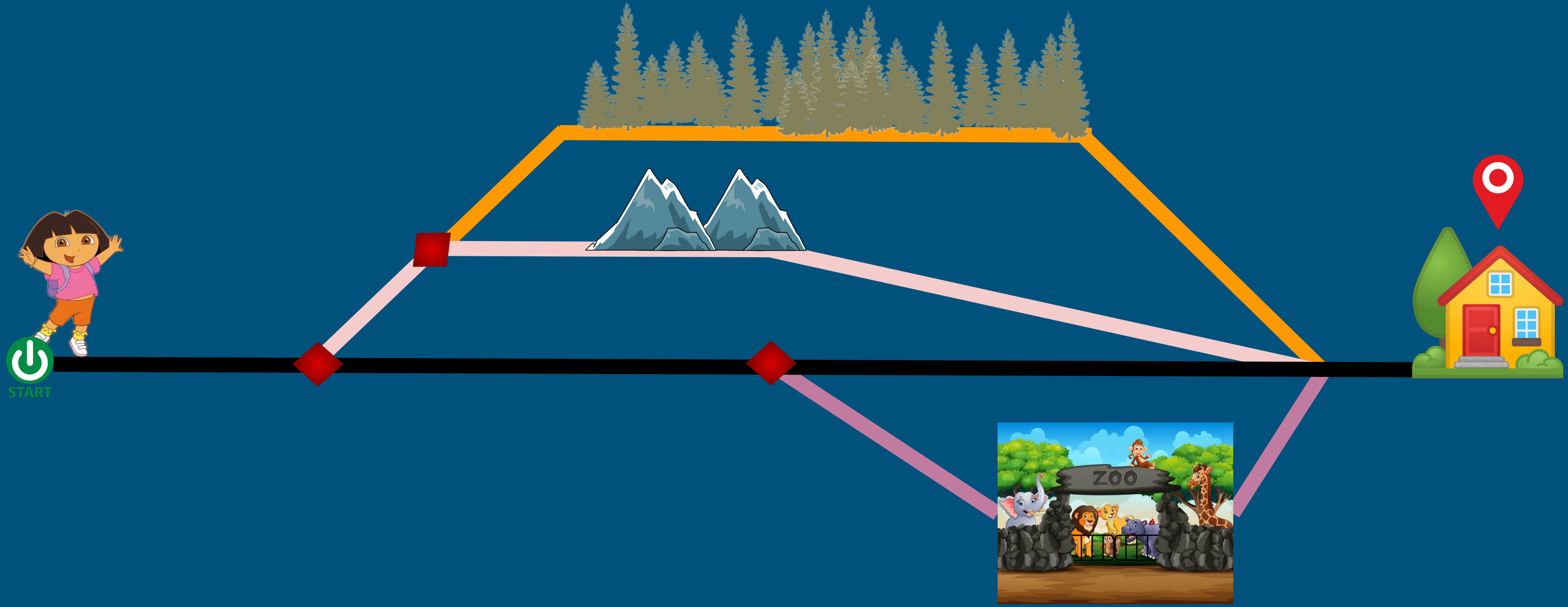


Law	and-major Form	or-major Form
Absorbtion	A and (A or B) ≡ A	A or (A and B) ≡ A
Associativity	A and (B and C) ≡ (A and B) and C	A or (B or C) ≡ (A or B) or C
Commutivity*	A and B ≡ B and A	A or B ≡ B or A
Complementarity	A and not A ≡ False	A or not A ≡ True
DeMorgan's Laws	not (A and B) ≡ not A or not B	not (A or B) ≡ not A and not B
Distributivity	A and (B or C) ≡ (A and B) or (A and C)	A or (B and C) ≡ (A or B) and (A or C)
Idempotence	A and A ≡ A	A or A ≡ A
Identity	A and True ≡ A	A or False ≡ A
Universal Bounds	A and False ≡ False	A or True ≡ True

Boolean Decisions

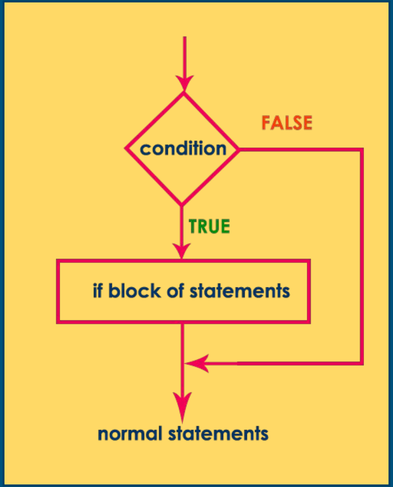


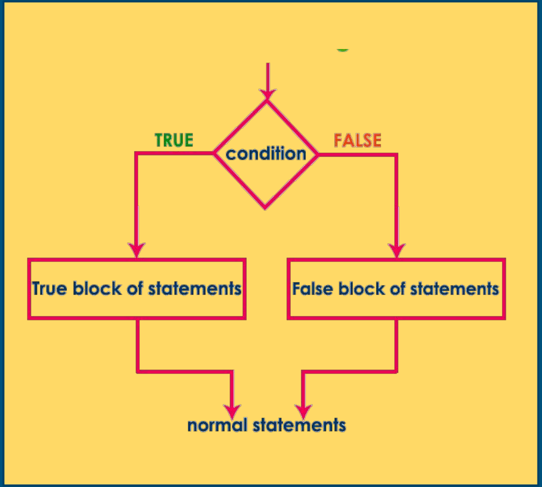
Branching/ Diversions

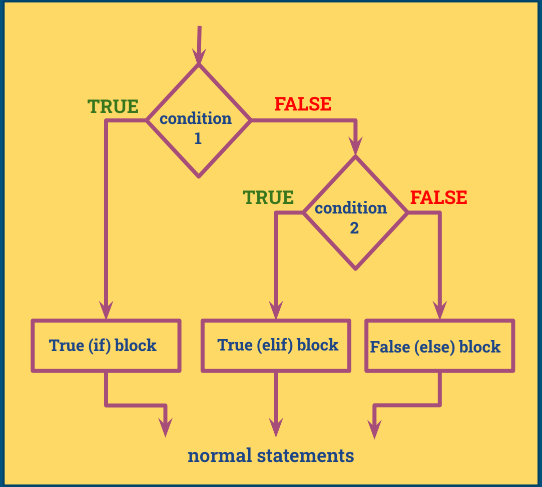


Dora's Exploration / Flow of Program Execution

Branching Statements (Conditionals)







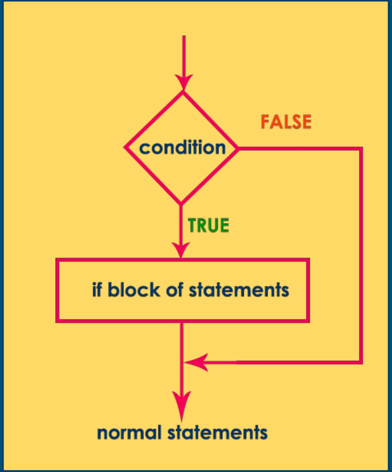
```
if condition:
        Statement_1
        Statement_2
        Statement_3
          ...
```

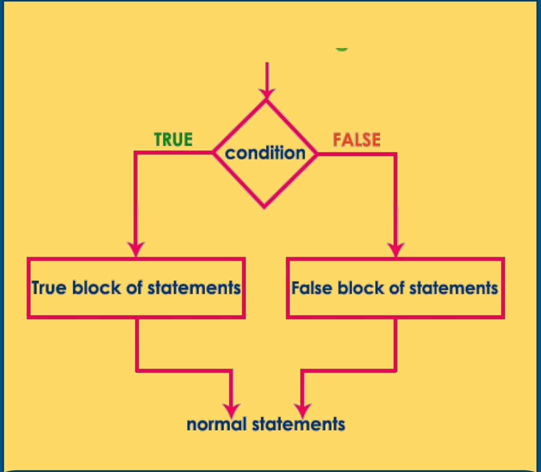
```
if condition:
        Statement_1
        Statement_2
        Statement_3
        ...
else:
        Statement_4
        Statement_5
        ...
```

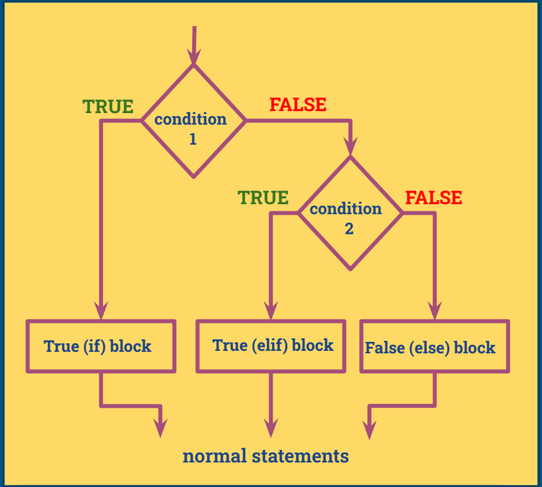
```
if condition_1:
        Statement_1
        Statement_2
        Statement_3
        ...
elif condition_2:
        Statement_4
        Statement_5
                ...
else:
        Statement_6
        Statement_7
        ...
```

→ 1 tab indentation

Branching Statements: Examples







```
answer = input(“Are you going out to play?”)

if answer == “yes”:
        print(“Put on a hat!”)
```

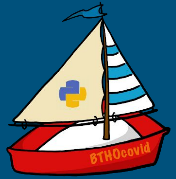
```
answer = input(“Is it raining?”)

if answer == “yes”:
        print(“Take an umbrella!”)
else:
        print(“Put on a hat!”)
```

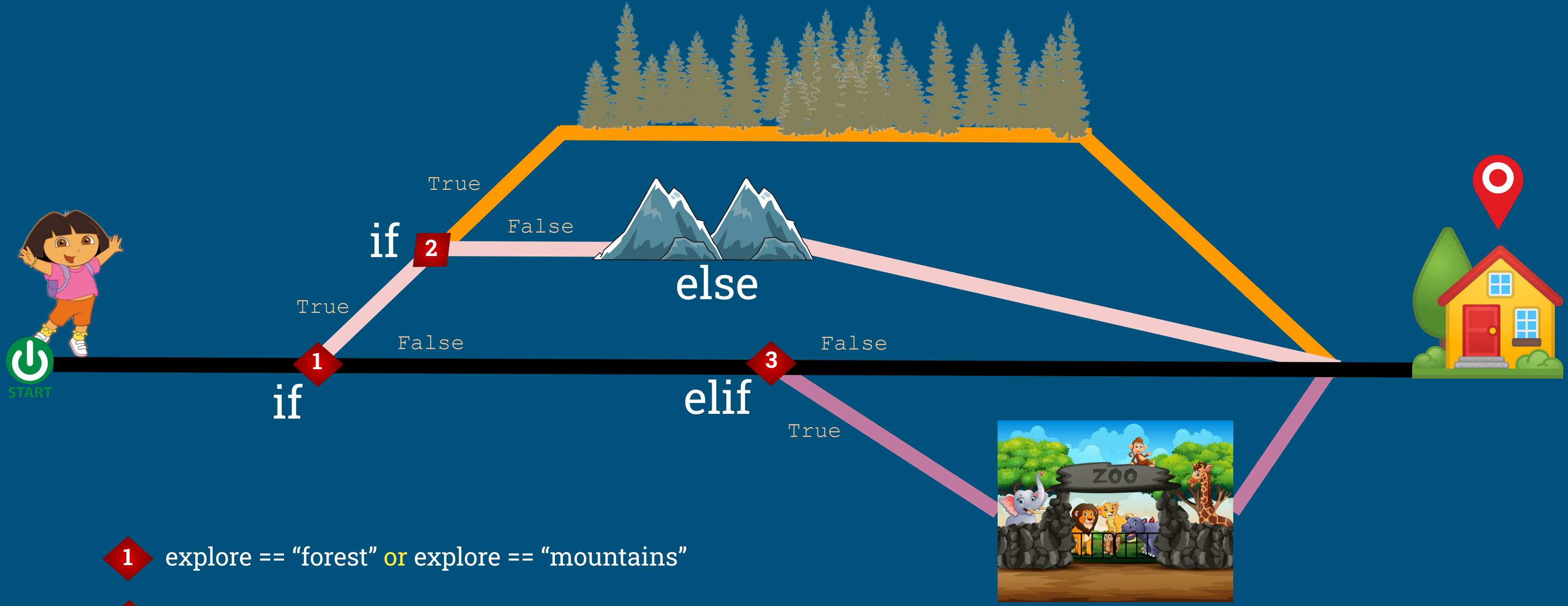
```
answer_1 = input(“Is it raining?”)
answer_2  = input(“Are  you  going  out  to  play?”)
if answer_1 == “yes”:
        print(“Take an umbrella!”)

elif answer_2 == “yes”:
        print(“Put on a hat!”)

else:   print(“Finish your homework!”)
```



Dora’s Exploration : conditionals



1 explore == “forest” or explore == “mountains”

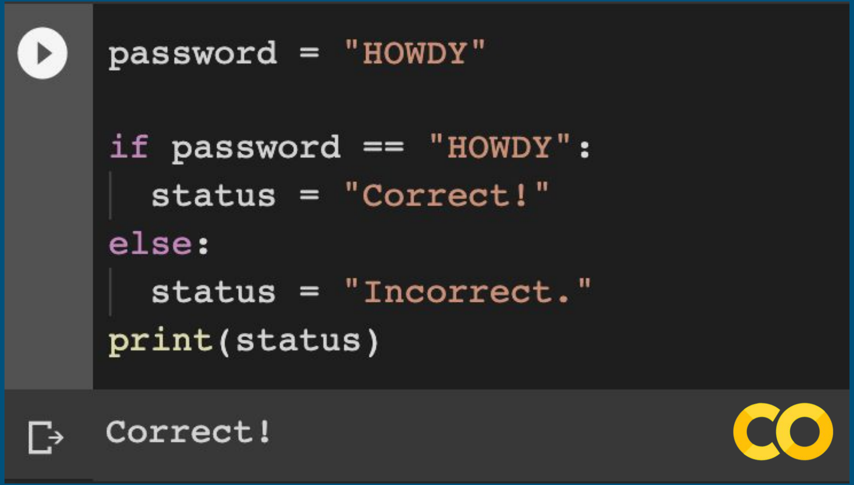
2 explore == “forest”

3 explore == “zoo”

Quickest way to reach Home is to **Not Explore**

Conditional Assignment

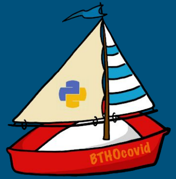
How can we represent these multiple assignment statements under conditionals in a concise way ?



```
password = "HOWDY"

if password == "HOWDY":
    status = "Correct!"
else:
    status = "Incorrect."
print(status)
```

```
Correct!
```

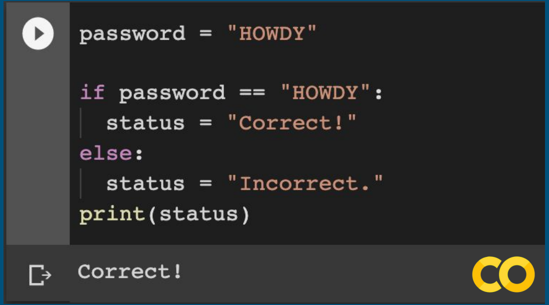


Conditional Assignment

A single assignment statement that evaluates different expressions based on a condition

```
if condition:
    x = true_value
else:
    x = false_value
```

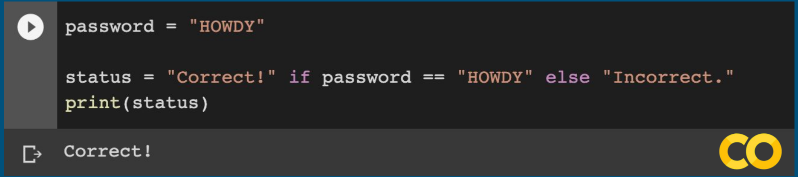
```
x = true_value if condition else false_value
```



```
password = "HOWDY"

if password == "HOWDY":
  status = "Correct!"
else:
  status = "Incorrect."
print(status)
```

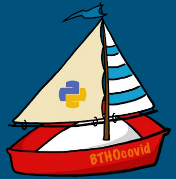
```
Correct!
```



```
password = "HOWDY"

status = "Correct!" if password == "HOWDY" else "Incorrect."
print(status)
```

```
Correct!
```



Next Lecture

L3: Repetitions (Loops)

Sat, April 25 (11 am-12 noon CDT)