Quiz II
Review Session

Topics List

1. Booleans
2. Logical Operators and Errors
3. Loops
4. Control Flow

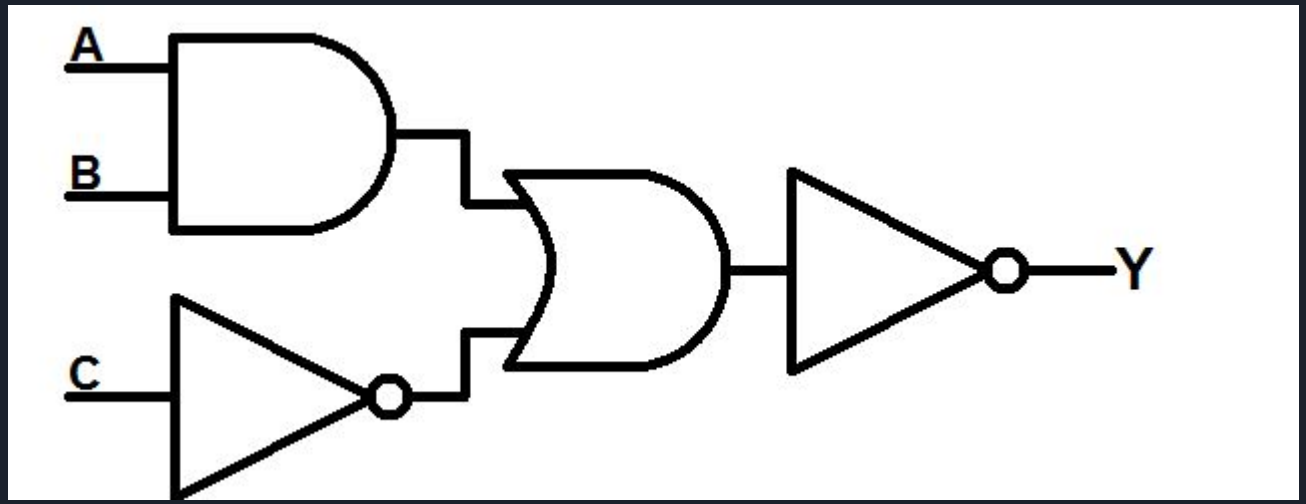
Booleans

1. Translate Boolean expressions to truth tables and circuits
2. Translate circuits to truth tables and Boolean expressions
3. Recognize how addition is done at the circuit level using algorithms and abstraction

Translating Boolean Expressions to Truth Tables and Circuits

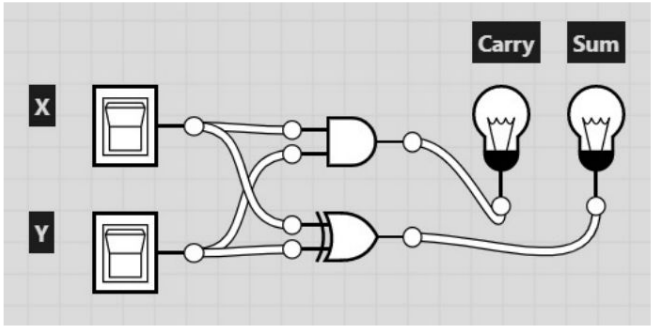
(x AND z) XOR (NOT (y OR x))

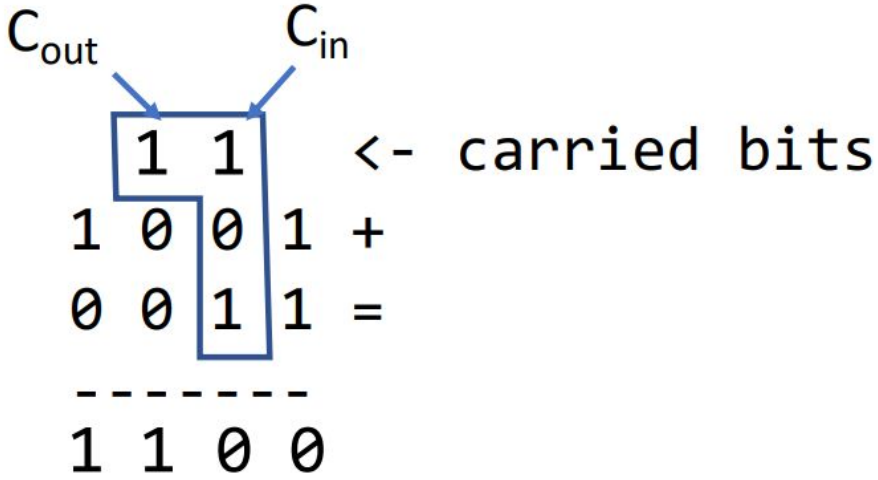
Translating Circuits to boolean expressions and Truth Tables



Addition at Circuit Level - Half Adder

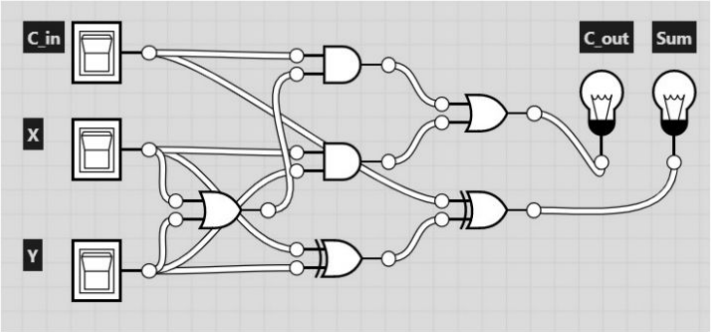
X	Y	X + Y	Carry	Sum	X ∧ Y	X ⊕ Y
1	1	10	1	0	1	0
1	0	01	0	1	0	1
0	1	01	0	1	0	1
0	0	00	0	0	0	0





Addition at Circuit Level - Full Adder

C_{in}	X	Y	C_{out}= $((X \vee Y) \wedge C_{in}) \vee (X \wedge Y)$	Sum = $(X \oplus Y) \oplus C_{in}$
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0



Logical Operators & Errors

1. Use logical operators on Booleans to compute whether an expression is True or False
2. Use conditionals when reading and writing algorithms that make choices based on data
3. Recognize the different types of errors that can be raised when you run Python code

Logical Operators

```
b1 = True
b2 = False

both = b1 and b2
either = b1 or b2

num1 = 5
num2 = 3

greater = num1 > num2
lesser = num1 < num2
equal = num1 == num2
notEqual = num1 != num2
```

```

short1 = b1 or num1 > num2
short2 = b2 and num1 < num2

s1 = "hello"
s2 = "e"
inside = s2 in s1
notInside = s2 not in s1
```

Conditionals on algorithms

Sometimes we want a program to do one of two alternative actions based on the condition. In this case, instead of writing two `if` statements, we can write a single `if` statement and add an **`else`**.

The `else` is executed when the Boolean expression is `False`.

```
if <BooleanExpression>:
    <bodyIfTrue>
else:
    <bodyIfFalse>
```

`if` clause

`else` clause

Conditionals on algorithms

Finally, we can use **`elif`** statements to add alternatives with their own conditions to `if` statements. An `elif` is like an `if`, except that it is checked **only if all previous conditions evaluate to `False`**.

```
if <BooleanExpressionA>:
    <bodyIfATrue>
elif <BooleanExpressionB>:
    <bodyIfAFalseAndBTrue>
else:
    <bodyIfBothFalse>
```

Types of Errors

```
1.) HellomyName Is = "Fernanda"
2.) favoriteFruit = "green"
3.) 34 = x
4.) two = 2
    two + 4
5.) print("I love taking 15-110')
6.) total = 150
    print("Your total is" + total)
```

Loops

1. Identify start values, continuing conditions, and update actions for loop control variables
2. Recognize which numbers will be produced by a range expression
3. Use for loops to loop over strings by index
4. Use for loops when reading and writing algorithms to repeat actions a specified number of times
5. Use while loops when reading and writing algorithms to repeat actions while a certain condition is met

For Loops

A **for loop** over a **range** tells the program exactly how many times to repeat an action. The loop control variable is updated by the loop itself!

```
for <loopVariable> in range(<maxNumPlusOne>):
    <loopBody>
```

```
for i in range(2, 11, 2):
    print(i)
```

Looping over strings

If the string is `s`, the string's first index is `0` and the last index is `len(s) - 1`. Use `range(len(s))`.

```
s = "Hello World"
for i in range(len(s)):
    print(i, s[i])
```

For Loops - Code Writing

For Loops

Write the function findSmiley(s), which takes in a string and returns whether or not there was a smiley face(“:)”) in that string. Your returned value should be a boolean and don’t use any fancy string methods to find the answer to this problem.

Code:

While Loops

A **while loop** is a type of loop that keeps repeating only while a certain condition is met. It uses the syntax:

```
while <booleanExpression>:
    <LoopBody>
```

Unlike `if` statements, the condition in a `while` loop **must eventually become `False`**. If this doesn't happen, the `while` loop will keep going forever!

While Loops - Code Writing

1. Write a function that given an integer n, from 0 to 100, it returns how many times we need to generate a number randomly to get n.

Control Flow Charts

1. Translate algorithms from control flow charts to Python code
2. Use nesting of statements to create complex control flow

Control Flow Charts