W05 LEC

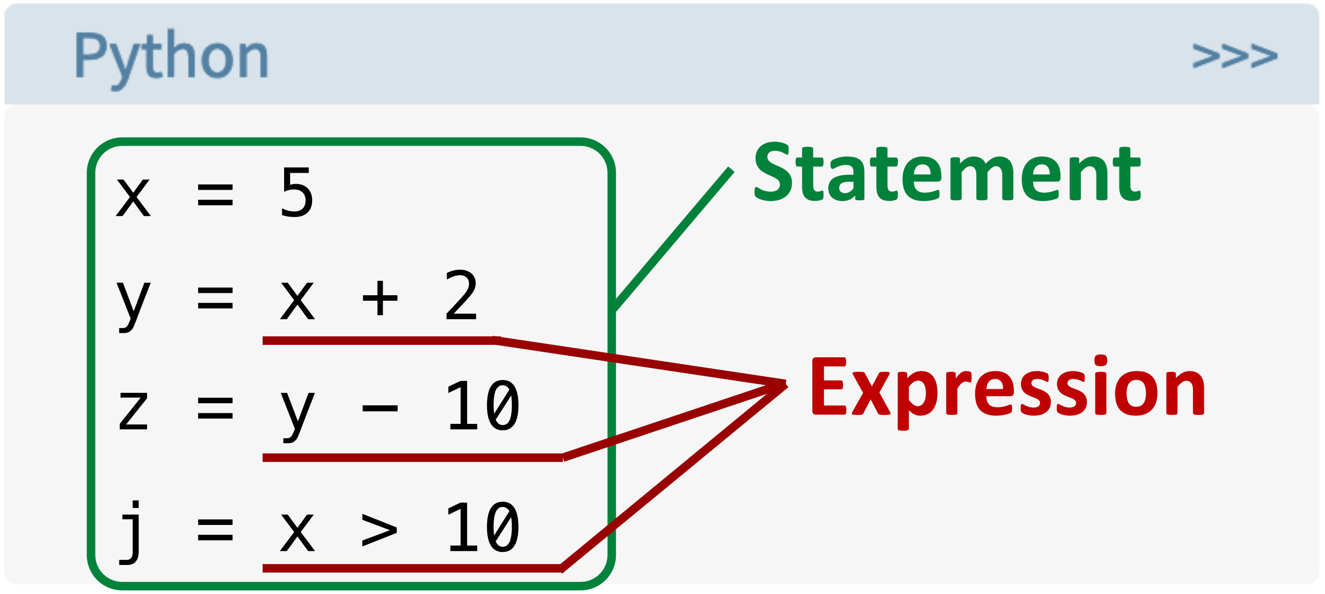
Conditionals Part I

Agenda

- Statement vs Expression
- Boolean
- Boolean Operators
- Conditionals
  - If statement
  - If-else statement
- Exercises

Statement vs Expression

- **Statement** is a line of code
- **Expression** is a piece of code that **creates a value** or **evaluates to a value.**



```
Python                                   >>>
x = 5
y = x + 2
z = y - 10
j = x > 10
```

Recap

- Three data types we have used so far are:
  - String (str) e.g., “Hello World”, “101”
  - Integer (int) e.g., 101
  - Floating-point (float), e.g., 3.14, 19.99

Today we are introducing a new data type:

Boolean

Boolean

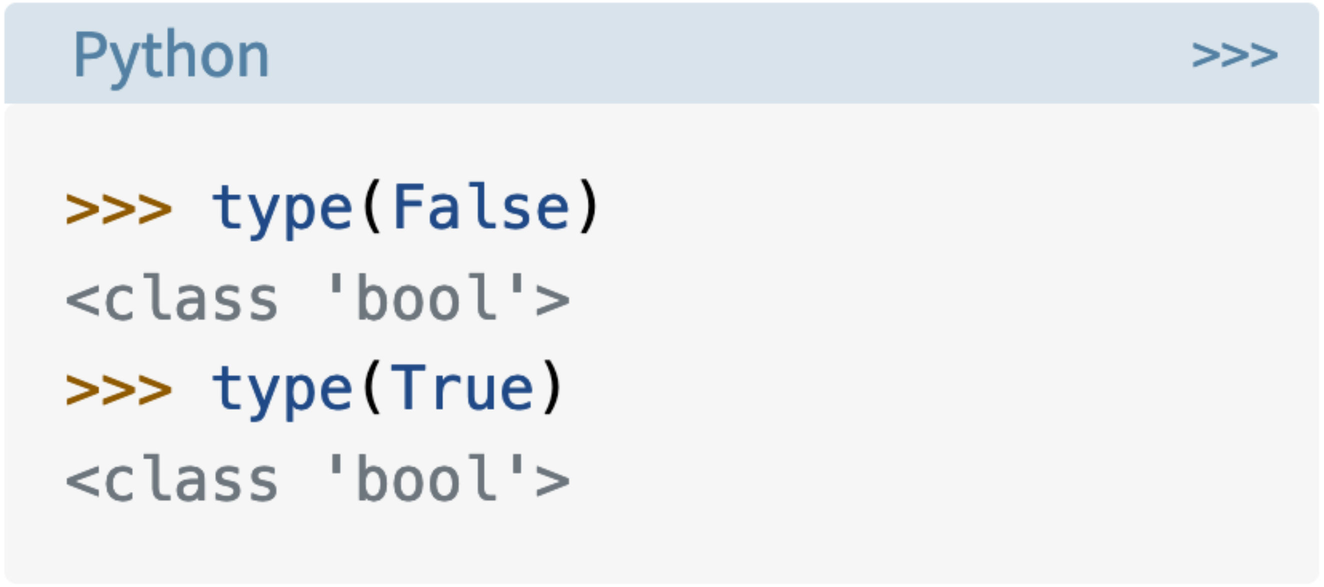
Sounds like “Boo-Lee-Uhn”

IPA /ˈbuli.ən/

Boolean Data Type

- It has two possible values:
  - **True**
  - **False**

Note that ***True*** and ***False*** are reserved keywords in Python.



```
>>> type(False)
<class 'bool'>
>>> type(True)
<class 'bool'>
```

Boolean Data Type

- Like the other 3 data types we've used earlier, it is possible to assign a Boolean value to variables.
- But it is not possible to assign a value to True and False (because they are reserved keywords)

```python
isHuman = True
print(isHuman)
print(type(isHuman))
```

Output:

```
True
<class 'bool'>
```

```python
True = 5
```

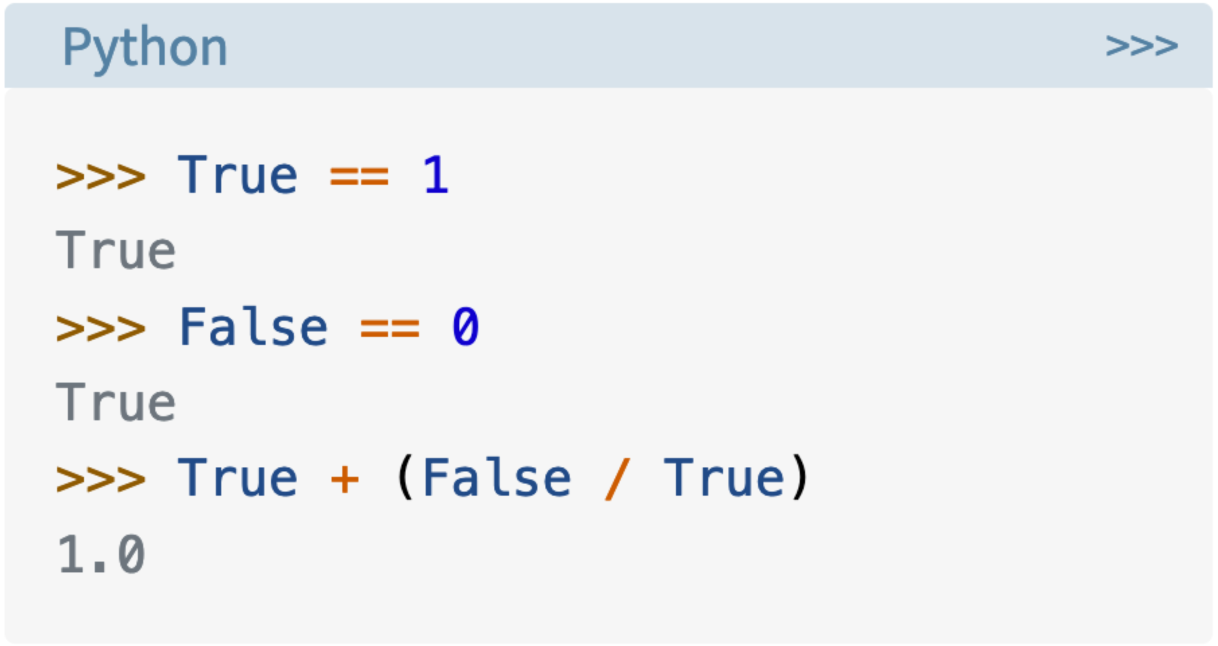
Gives you a SyntaxError

Boolean as Numbers

- Boolean are considered a **numeric** type in Python. This means they are numbers for all intents and purposes.

True is 1

False is 0



```
>>> True == 1
True
>>> False == 0
True
>>> True + (False / True)
1.0
```

The bool() function

- Like the int() and float() functions, the bool() function converts a value to the corresponding Boolean value.

```
bool(1)
# result = true
```

```
bool(0)
# result = false
```

```
bool("1")
# result = true
```

```
bool(6.66)
# result = true
```

Zero, in any numeric type such as int and float (e.g., 0, 0.0), will be evaluated to False. all other values will be True

```
bool("false")
# result = false
```

```
bool("f")
# result = false
```

```
bool("true")
# result = true
```

```
bool("beneath the remains")
# result = true
```

Boolean Operators

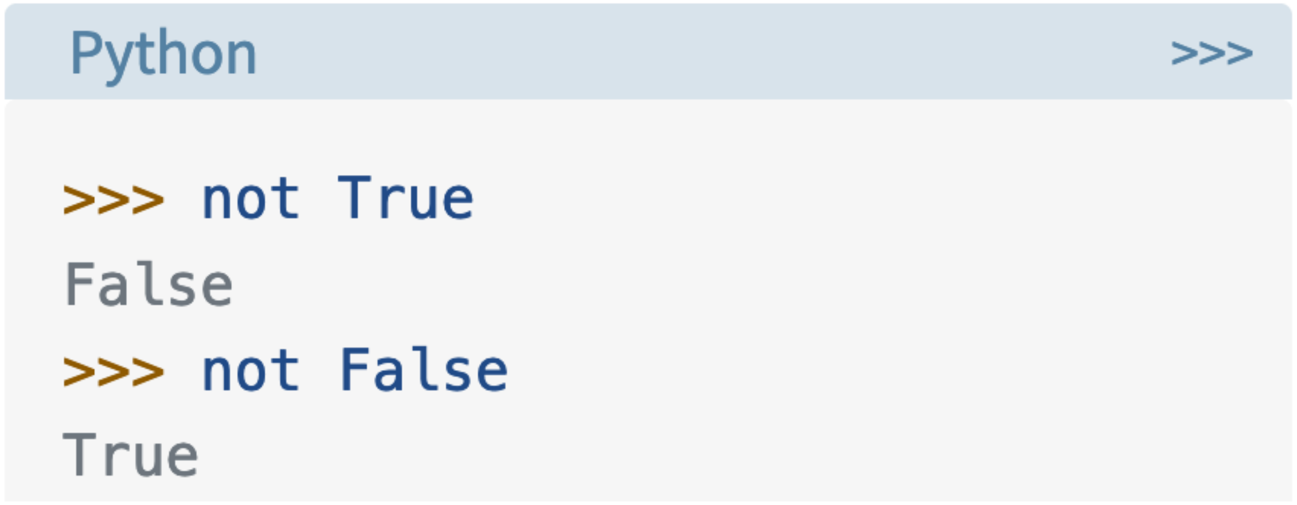
Boolean Operators

- Boolean operators are those that take **Boolean inputs** and return **Boolean results.**

  - The “**not**” operator
  - The “**and**” operator
  - The “**or**” operator

The “not” operator

- This operator takes one argument and returns the opposite result.
- The “not” keyword is placed before Boolean variables/expressions.

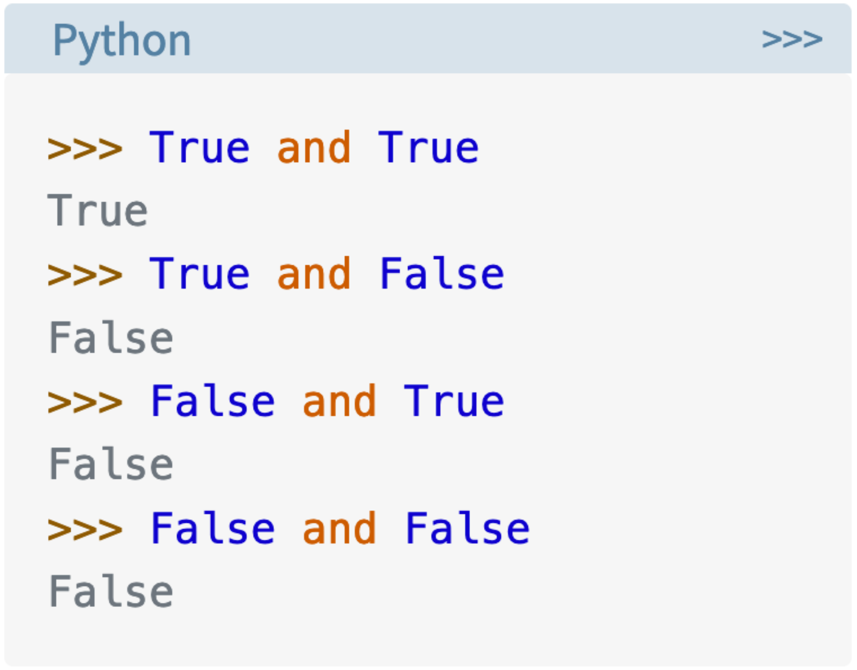


```
>>> not True
False
>>> not False
True
```

p	~p
T	F
F	T

The “and” operator

- This operator takes two arguments.
- It only evaluates to True if both are True, otherwise it evaluates to False



Python >>>

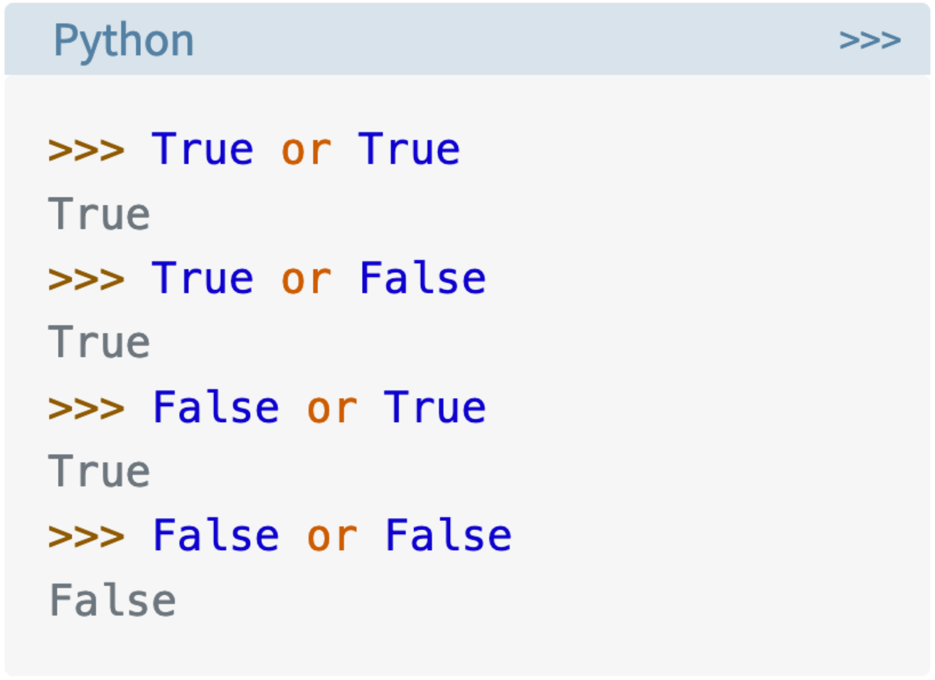
```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

p	q	p∧q
T	T	T
T	F	F
F	T	F
F	F	F

The “or” operator

- This operator takes two arguments.
- It evaluates to True if one of the two arguments is True, or both are True.

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F


Python >>>

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

Comparison/Relational Operators

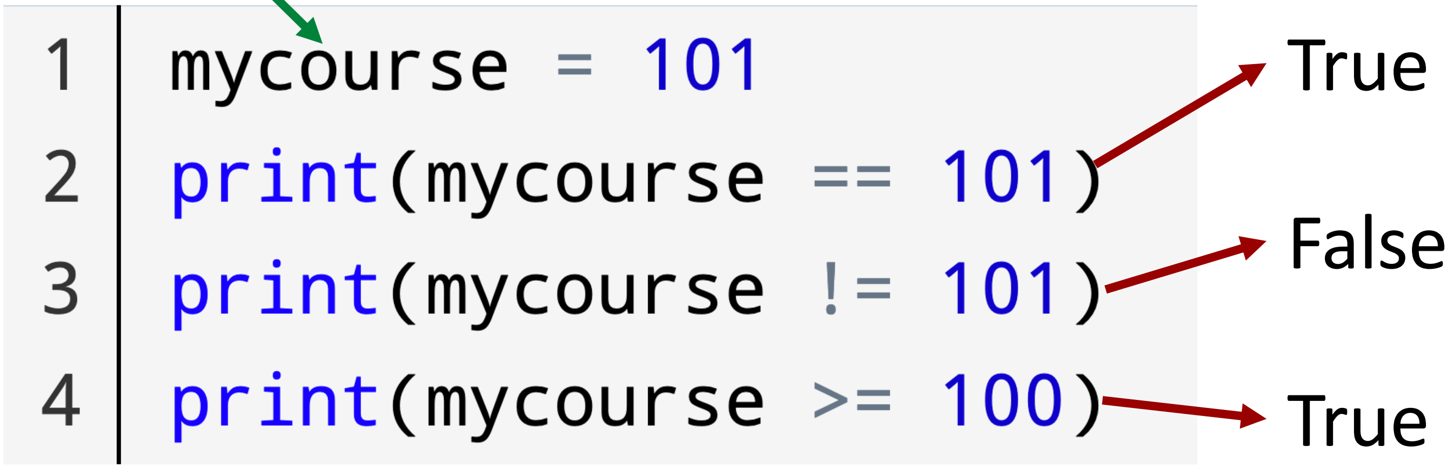
- Equality (**==**):
  - Checks if two values are equal.
- Inequality (**!=**):
  - Checks if two values are not equal.
- Greater than (**>**)
  - Checks if one value is greater than another.
- Less than (**<**)
  - Checks if one value is less than another.
- Greater than or equal to (**>=**)
  - Checks if one value is greater than or equal to another.
- Less than or equal to (**<=**)
  - Checks if one value is less than or equal to another.

These operators also take two arguments and **returns a Boolean**. These work for all 3 data types (string, float, int) we've learned so far

Example 1:

- The Boolean values ***True*** and ***False*** are returned when an expression is compared or evaluated with Comparison/Relational operators

Assigning 101 (int) to a variable named "mycourse"



```python
mycourse = 101
print(mycourse == 101)
print(mycourse != 101)
print(mycourse >= 100)
```

```
True
False
True
```

Comparing the value stored in mycourse to an integer

Example 2:

- Let **a = 10** and **b = 20**

Comparing values stored in a and b in many different ways

Operator	Description	Example	Result
`==`	`equal`	`a == b`	🟩
`!=`	`not equal`	`a != b`	🟩
`>`	`greater than`	`a > b`	🟩
`<`	`less than`	`a < b`	🟩
`>=`	`greater than or equal`	`a >= b`	🟩
`<=`	`less than or equal`	`a <= b`	🟩

Combining Expressions

- You can then combine multiple expressions that evaluate to a Boolean to create more complex expressions

```
# Variables
temperature = 22
weather = "sunny"

# Multiple expressions that evaluate to a Boolean
is_warm = temperature > 20
is_sunny = weather == "sunny"

# Combine the expressions
should_picnic = is_warm and is_sunny

print(should_picnic) # Outputs: True
```

Check Your Understanding

Can you guess the outputs of the following code?

```
x = 10
y = 2
z = "hello101"
print(x == 5 or y < 5)
print((not x < 4) and y > 1)
print(x == 10 and y == 2)
print(z == "hello101" and y == 2)
print(z != "hello101" and y >= 2)
```

Be Careful

- It is easy to accidentally write an expression that is always True (***tautology***) or always False (***contradiction***)

What's wrong with these expressions?

```
count >= 10 or count < 50

count < 10 and count > 100
```

Conditionals

Control Structure

- So far we have only seen **sequential execution** (statements are performed one after the other)
- However, we may need to skip over some statements and perform some other statements based on some condition
- Ex. *"if it rains, I skip the class."*
- Ex. *"if I'm younger than 20 years old, I'm not legal to drink"*

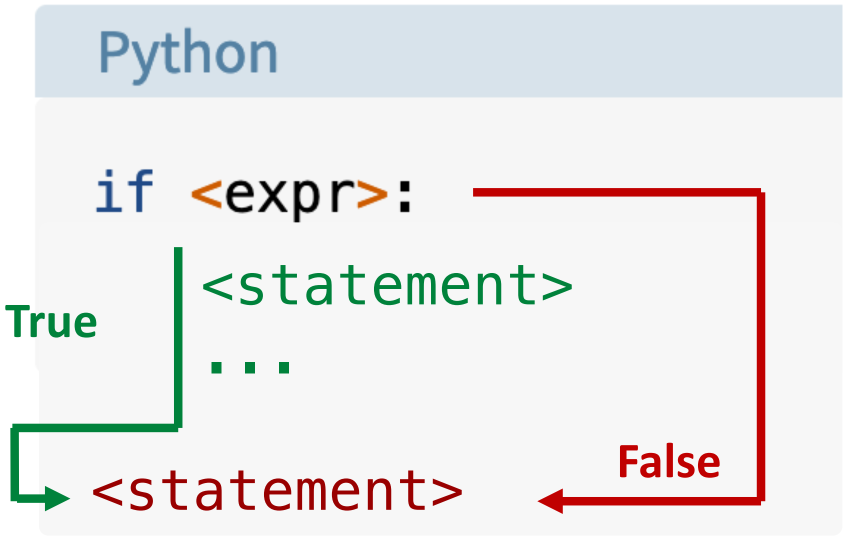
Introducing

Control Structures

Control Structures

- A control structure directs the order of execution of the statements in a program.
- One of the basic control structures:

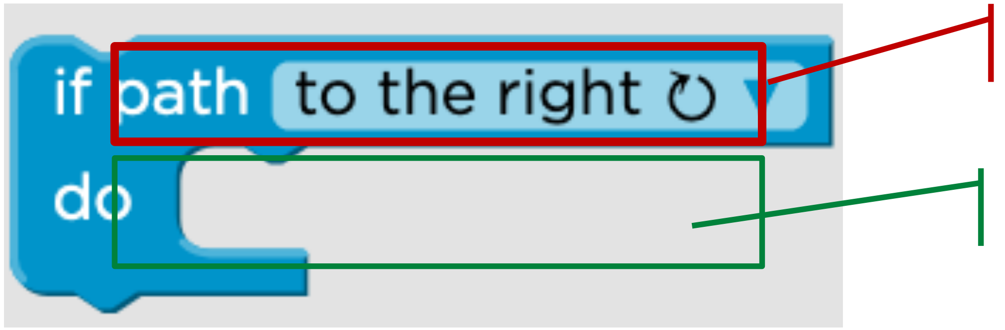
the `if` statement



```
if <expr>:
    <statement>
    ...
<statement>
```

- **<expr>** - an expression evaluated to a Boolean value we saw previously.
- **<statement>** a block of statements (**note the identation)** that will be executed when the <expr> is evaluated to **True** or skip over all **<statements>** if **False**
- Execution will then proceed with <statement> either way

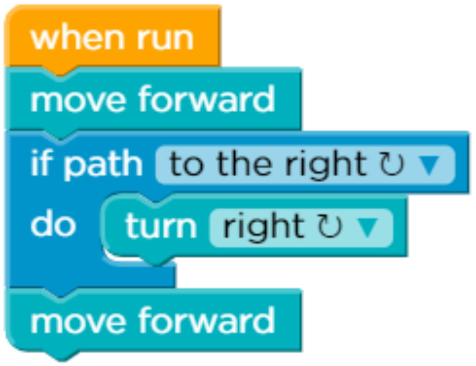
Remember the Maze Game?



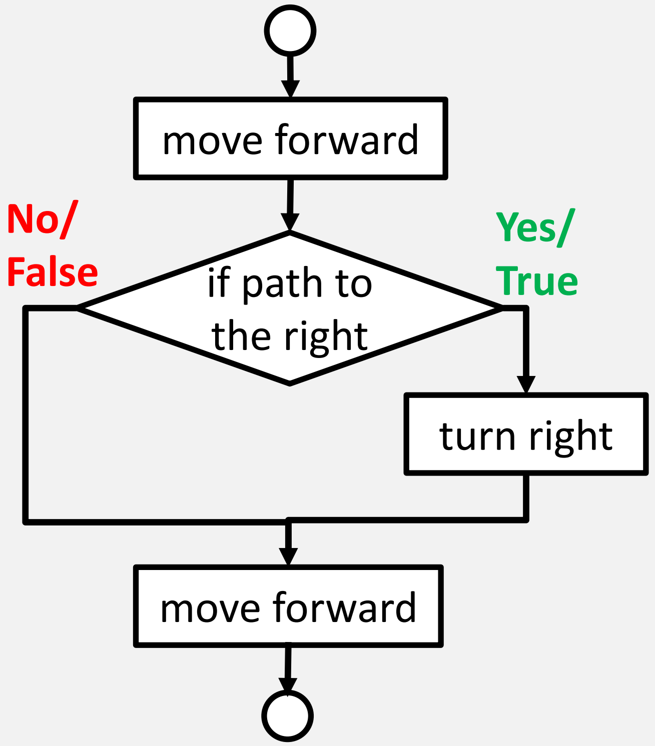
Evaluate this expression

Do this when the **expression** is True

Block-based View



Flow Chart View



Python Code View

```
move_forward()
if  path_to_the_right:
      turn_right()
move_forward()
```

Check Your Understanding

Can you guess the output?

```
x = 10
if x == 10:
    print("the expr is True")
    print("this block is executed")

print("after conditional")
```

Answer

```
x = 10
if x != 10:
    print("the expr is True")
    print("this block is executed")

print("after conditional")
```

Answer

Coding Excercise

- Write a program to prompt the user for a number
- If the user enters any positive number, print "*Good job!*" *and* "*You guessed it right*" on two separately lines
- Regardless of whether the user guessed the number correctly, the program should print "*Thanks for playing the game!*"

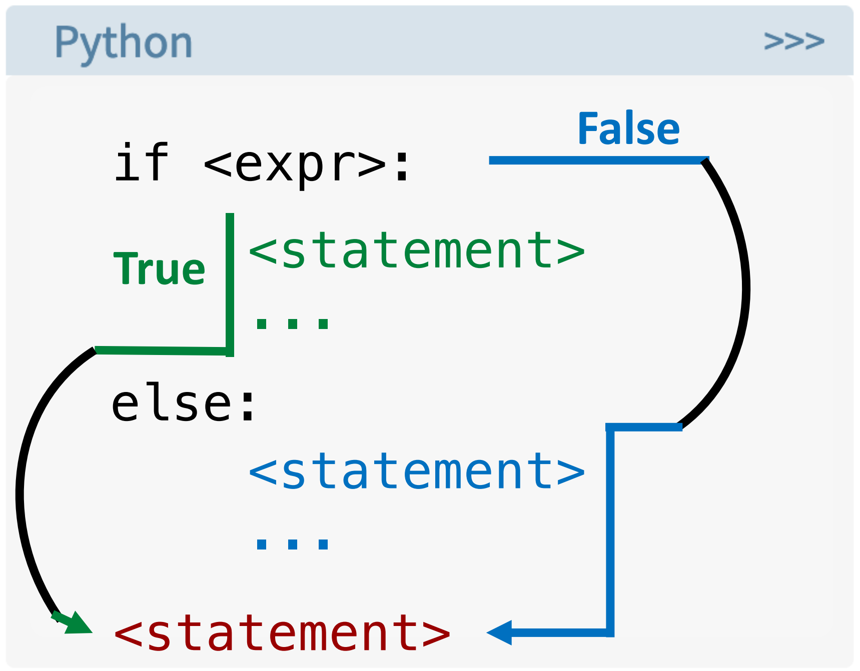
Coding Exercise - Solution

Solution

A More Advanced Control Structure

Sometimes, you want to evaluate a condition and take on path if its true but specify an alternative path if it is not.

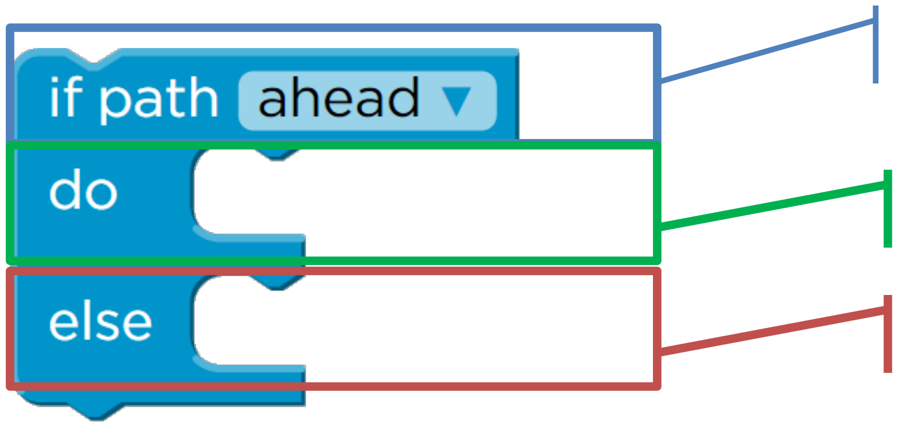
The **if-else** statement



```
if <expr>:
    <statement>
    ...
else:
    <statement>
    ...
<statement>
```

- **<expr>** - an expression evaluated to a Boolean value we saw previously.
- **<statement>** a block of statements that will be executed if <expr> is evaluated to **True**
- **<statement>** a block of statements that will be executed if <expr> is **False**
- Execution will then proceed with **<statement>** either way

Remember the Maze Game?

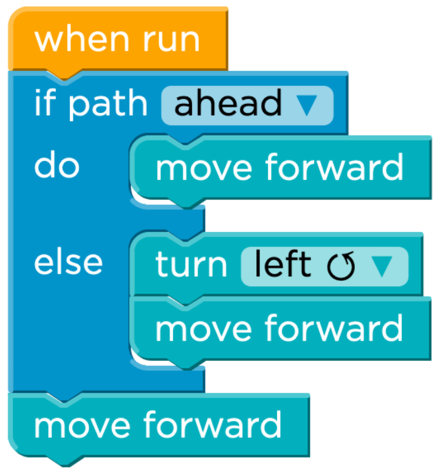


Evaluate this expression

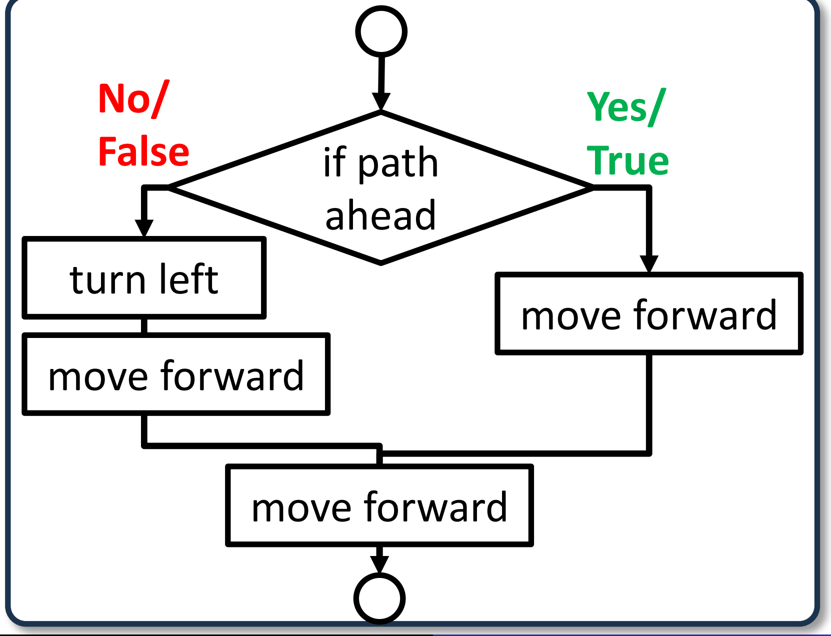
Do this when the expression is True

Do this when the expression is False

Block-based View



Flow Chart View

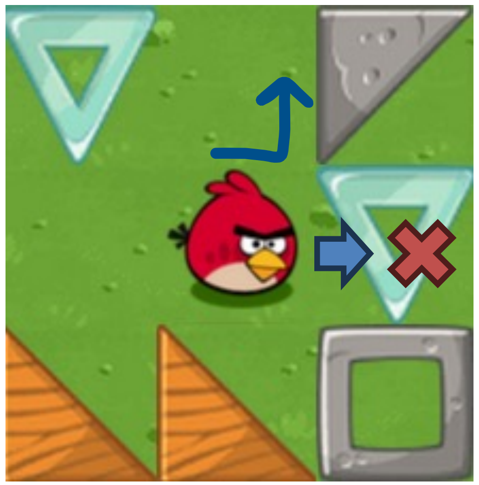


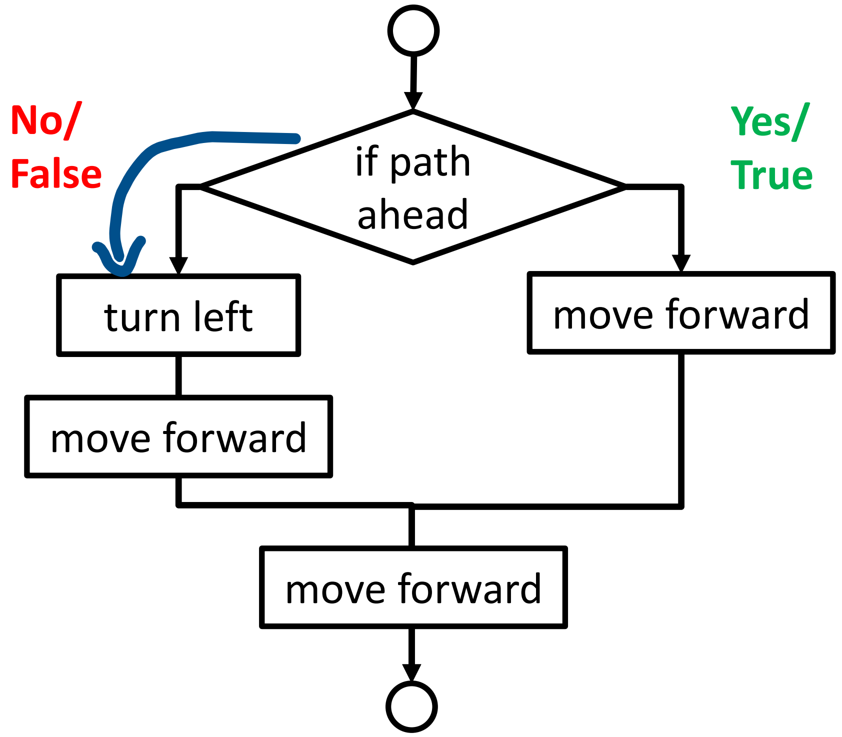
Python Code View

```
if  path_ahead:
     move_forward()
else:
     turn_left()
     move_forward()
move_forward()
```

Selecting a Path

- Once a path is selected, the code/blocks on the other path will not be executed
- Unless looping is involved...





Real Life Scenario

If the weather is "nice"

- Take my dog for a walk
- Ride a bicycle
- Go grocery shopping

else

- Stay at home
- Watch Netflix Movies/TV Shows
- Have a facetime call with friends

Regardless, **I'll have dinner at home**

Real Life Scenario

If the weather is "nice"

- Take my dog for a walk
- Ride a bicycle
- Go grocery shopping

else

- Stay at home
- Watch Netflix Movies/TV Shows
- Have a facetime call with friends

Regardless, I'll **have dinner at home**

Let's turn this into code

(print out each action on separate lines)

One requirement though

You have to prompt the user to enter the condition of the weather.

(Hint: see how you can compare strings in Slide #19)

Real Life Scenario to Code

Solution

Coding Exercises

Exercise – 1 (Clubbing)

As a bouncer at a club, your task involves deciding who can enter. Specifically, you need to verify if a person is at least 21 years old. If they meet this age requirement, you allow them entry. If not, you deny them entry.

Your program should ask the user for their age.

Exercise – 2 (Finding the min value)

Write a function named "*find_min*" that takes in two numbers as arguments and return the minimum of the two.

(The function should not have any print statement)

You should ask the user to input the two numbers

Exercise – 3 (Am I eligible to vote?)

Write a program that asks the user for their age and asks whether they are a citizen of the country ("yes" or "no").

If the user is 18 years old or older and they are a citizen, the program should print "You are allowed to vote." Otherwise, the program should print "You are not allowed to vote."

References:

- https://realpython.com/python-conditional-statements/
- https://realpython.com/python-boolean/