Decisions: If Else Statements

CPSC 217: Introduction to Computer Science for Multidisciplinary Studies I
July 2024

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

June 4, 2024

UNIVERSITY OF CALGARY

Conditions in Python

- Decision making/branching constructs in Python:
  - If
  - If-else
  - If-elif-else

Conditions in Python

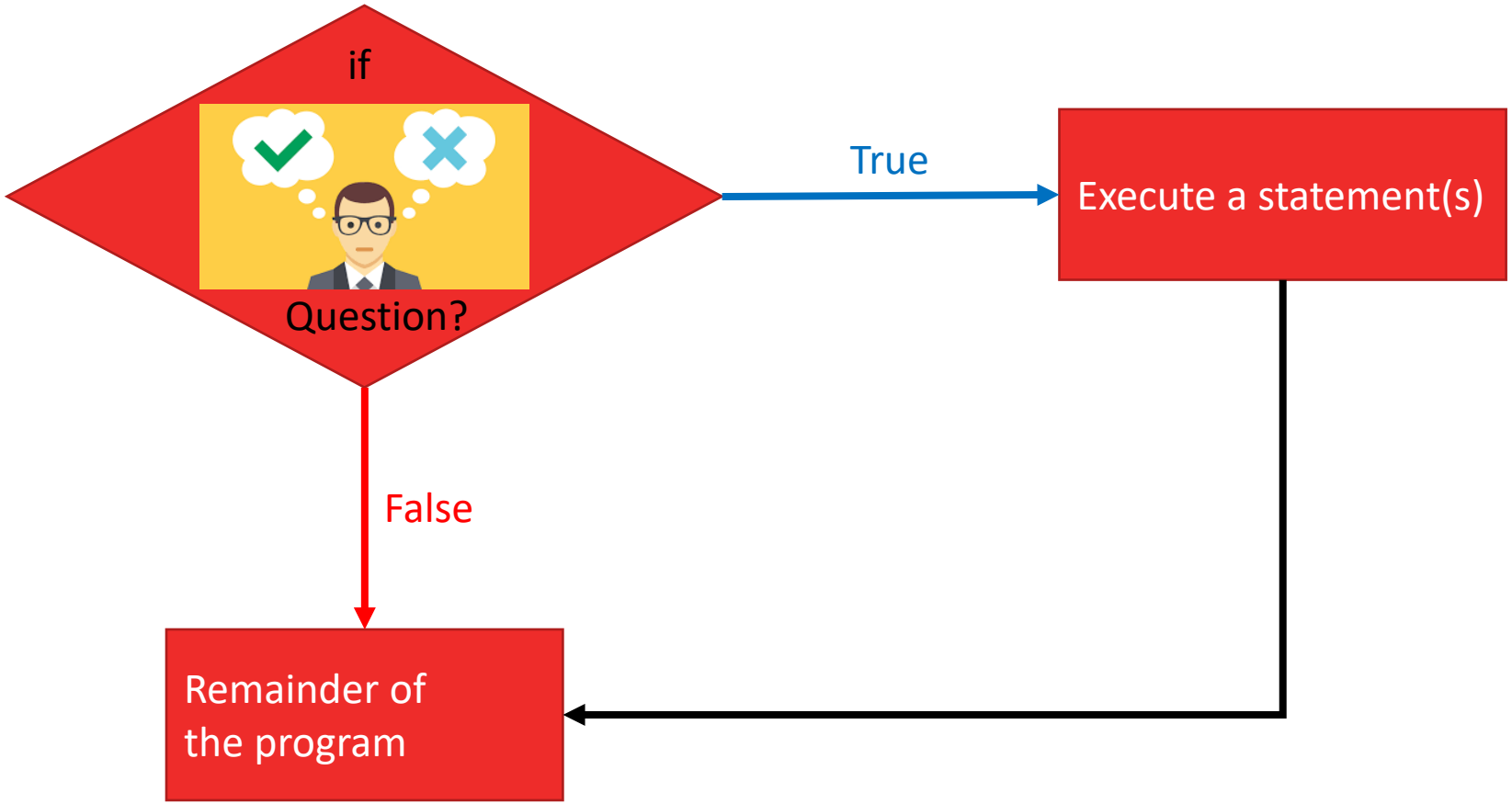
- Decision making/branching constructs in Python:
  - If
  - If-else
  - If-elif-else

  - If-elif
  - If-elif-elif…-elif

  - If-elif-else
  - If-elif-elif…-elif-else

if

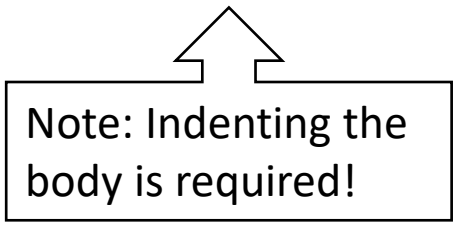
Condition flowchart



“if” statement format

- The logical expression is evaluated
  - If True, the indented statements that follow are executed
  - If False, the indented statements that follow will not be executed
- The program continues

if (*logical expression*):
    Body



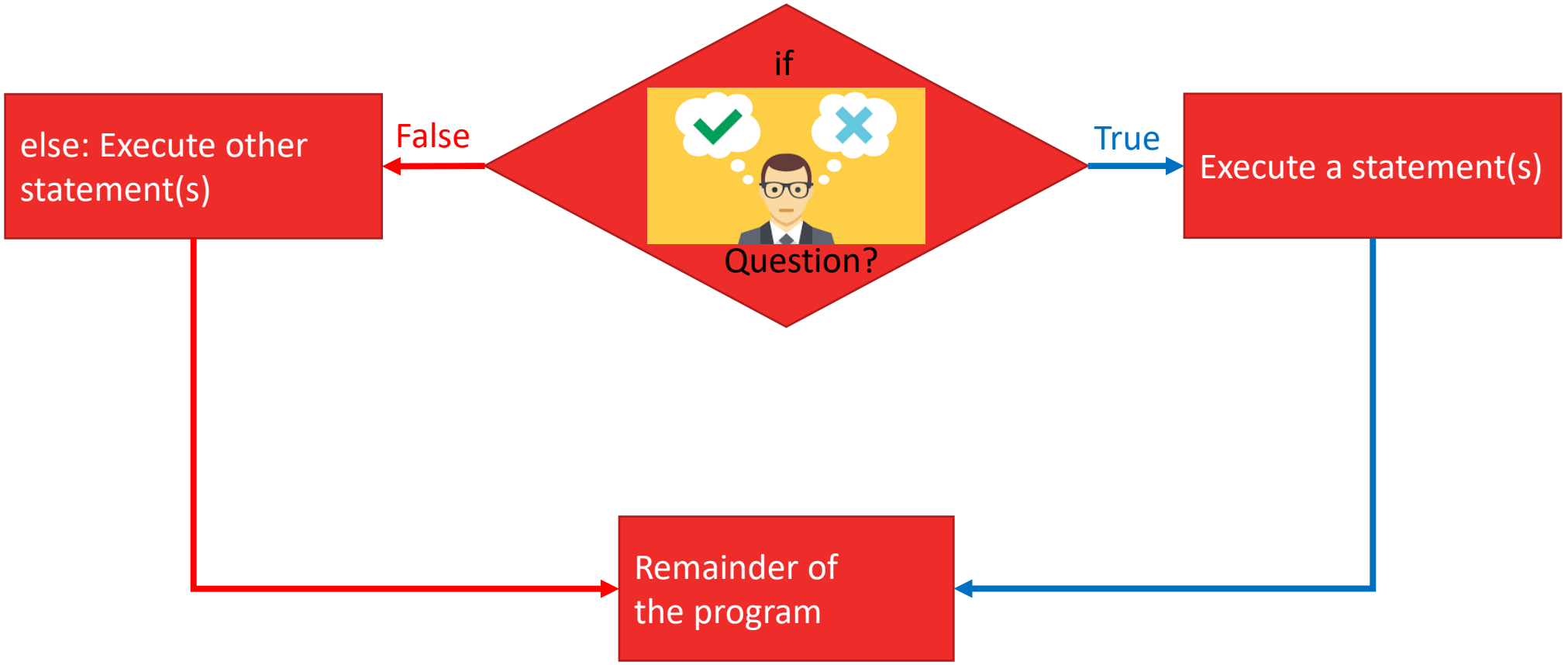

```
if age <= 17:
    print ('You are underage!')
```

Using if

```
#Start in middle
pointer.up()
pointer.goto(WIDTH/2,HEIGHT/2)

#Asking use for desired data
sXLocation = input("Enter new x coordinate in (800,600) window: ")
sYLocation = input("Enter new y coordinate in (800,600) window: ")
sColor = input("Enter color [1:red otherwise:default]: ")

x = int(sXLocation)
y = int(sYLocation)

if sColor == "1":
        pointer.color("red")

pointer.down()
pointer.goto(x,y)
pointer.up()

#Close the graphic window on user's click
screen.exitonclick()
```

else

If-Else statement



if-else format

if (*logical expression*)**:**

    body of 'if'

else:

    body of 'else'

remainder of the program

Example for if-else

```
if (grade >= 50):
        letterGrade = 'P'
        print ('You pass!')
else:
        letterGrade = 'F'
        print (‘You did not pass.')

print ('Thank You!')
```

Using if-else

```
#Start in middle
pointer.up()
pointer.goto(WIDTH/2,HEIGHT/2)

#Asking use for desired data
sXLocation = input("Enter new x coordinate in (800,600) window: ")
sYLocation = input("Enter new y coordinate in (800,600) window: ")
sColor = input("Enter color [1:red otherwise:blue]: ")

x = int(sXLocation)
y = int(sYLocation)

if sColor == "1":
        pointer.color("red")
else:
        pointer.color("blue")

pointer.down()
pointer.goto(x,y)
pointer.up()

#Close the graphic window on user's click
screen.exitonclick()
```

elif

else if

if-elif-else format

if (*logical expression*)**:**

    body of 'if'

elif *(logical expression)**:***

    body of 'elif'

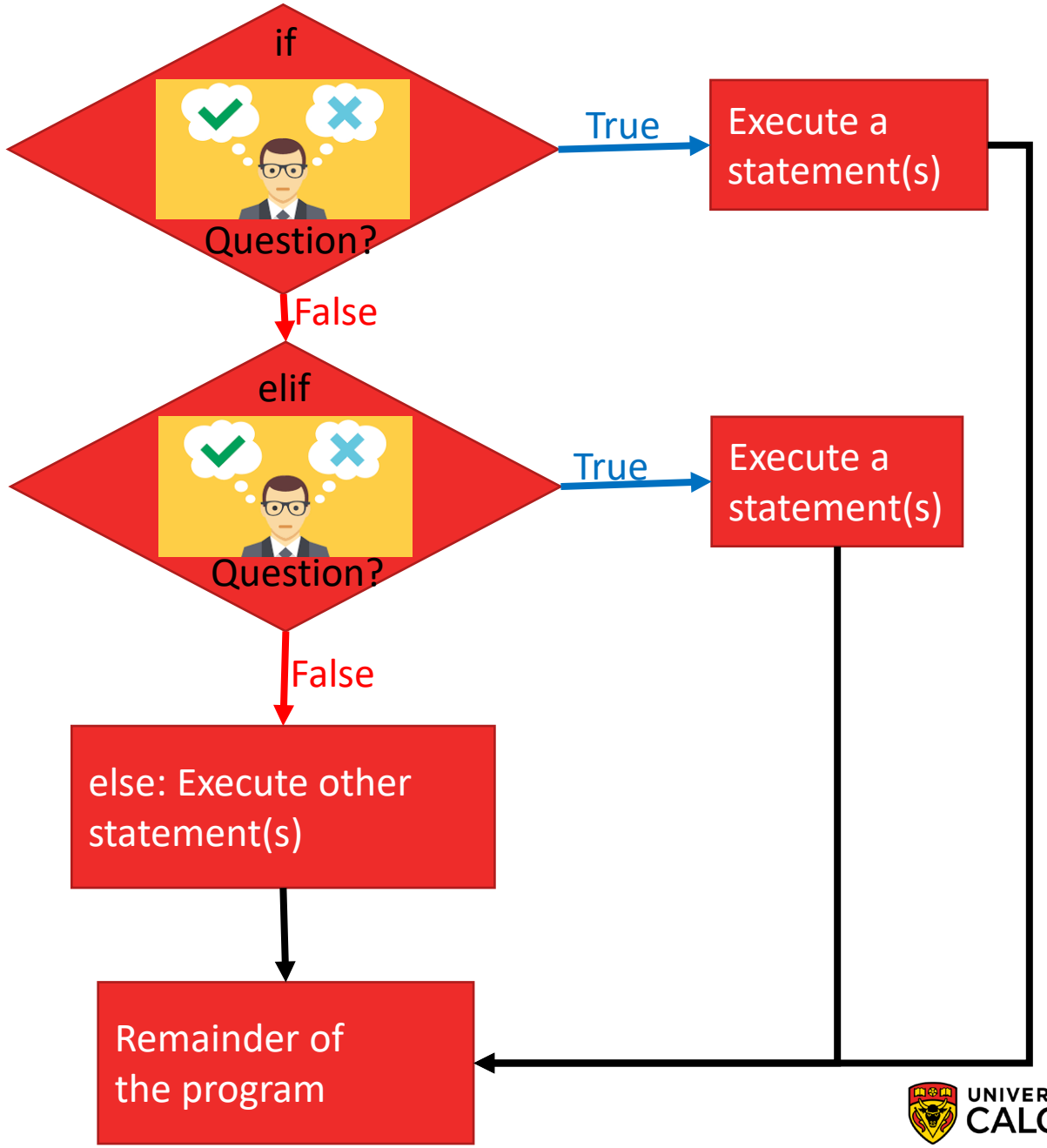
else:

    body of 'else'

remainder of the program

if –elif-else

```
x = 10
y = 10

if x < y:
    print("x is less than y")
elif x > y:
    print("x is greater than y")
else:
    print("x and y must be equal")
```



Using elif

```
if (grade >= 95):
    letterGrade = 'A+'
if (grade >= 90):
    letterGrade = 'A'
if (grade >= 85):
    letterGrade = 'A-'
if (grade >= 80):
    letterGrade = 'B+'
.
.
.
print (letterGrade)
```

What is the output if grade is 97? Multiple ‘ifs’ may not work for some problems.

Using elif

```
if (grade >= 95):
    letterGrade = 'A+'
if (grade >= 90):
    letterGrade = 'A'
if (grade >= 85):
    letterGrade = 'A-'
if (grade >= 80):
    letterGrade = 'B+'
.
.
.
print (letterGrade)
```

What is the output if grade is 97? Multiple ‘ifs’ may not work for some problems.

```
if (grade >= 95):
    letterGrade = 'A+'
elif (grade >= 90):
    letterGrade = 'A'
elif (grade >= 85):
    letterGrade = 'A-'
elif (grade >= 80):
    letterGrade = 'B+'
.
.
.
print (letterGrade)
```

if-elif-else provides the result.

Using if elif else

```
#Start in middle
pointer.up()
pointer.goto(WIDTH/2,HEIGHT/2)

#Asking use for desired data
sXLocation = input("Enter new x coordinate in (800,600) window: ")
sYLocation = input("Enter new y coordinate in (800,600) window: ")
sColor = input("Enter color [1:red 2:green 3:blue otherwise:black]: ")

x = int(sXLocation)
y = int(sYLocation)

if sColor == "1":
        pointer.color("red")
elif sColor == "2":
        pointer.color("green")
elif sColor == "3":
        pointer.color("blue")
else:
        pointer.color("black")

pointer.down()
pointer.goto(x,y)
pointer.up()

#Close the graphic window on user's click
screen.exitonclick()
```

if-elif-else

You can always have an ‘if’ without ‘elif’ or ‘else’

You can’t have ‘elif’ without ‘if’ first

You can’t have ‘else’ with ‘if’ or ‘elif’ ahead

You can only have one ‘if’ (in a chain)

You can only have one ‘else’ (in a chain)

You can have as many ‘elif’ as you desire (in middle of chain)

Nesting

Nesting

```
age = input("Enter an age:")
age = int(age)

print("--before--")

if(age > 0):
    print("you are alive")
    if(age <= 17):
        print("you are young")
    elif(age == 18):
        print("congratulations you are 18")
    else:
        print("you are olds")
        if(age >= 100):
            print("you are a centenarian")
else:
    print("you aren't alive")

print("--after--")
```

Scope

Scope

Namespace - is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc.) inside it.

Whenever you declare a variable it exists within a namespace

- Whenever python sees you use a variable is looks with this storage area for the variable **name** you use and finds out what it is attached to

dir() returns to use all the things in the current namespace

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>> x = 2
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'x']
>>>
```

Scope

The variables of

a = 2 , b = 3 , B = 4 , this = 10 , avacodo = 42

Would create variables of name **a,b,B,this,avocado** in the namespace (all of these will have the type int)

Python decides types for us (implicitly) based on what we put in

We can delete things from it using **del <name>**

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>> x = 2
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'x']
>>> del x
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
>>>
```

Scope

If we use a variable name that doesn't exist

```
print(dir())
print("x=1")
x = 1
print(dir())
print("del x")
del x
print(dir())
print(x)
```

Scope

If we use a variable name that doesn't exist

```
print(dir())
print("x=1")
x = 1
print(dir())
print("del x")
del x
print(dir())
print(x)
```

```
Traceback (most recent call last):
  File "C:/Users/jonat/Dropbox/CPSC231F19/scope2.py", line 8, in <module>
    print(x)
NameError: name 'x' is not defined
```

Scope

When we enter an indented block

- This namespace can be changed
- **But we have to be careful to not create a variable SOMETIMES**
- **That our code uses ALL the time**

```
age = input("Enter an age:")
age = int(age)


print("namespace before:", dir())

if (age > 0 ):
    print("namespace in if before:", dir())
    result = 2
    print("namespace in if after:", dir())

print("namespace after:", dir())
print(result)
```

Testing

Testing

- **Black-box testing**
  - **Test the program without looking at the source code**
  - **Tests are generally functional / behavioural**

- **White-box testing**
  - **Design test cases for the program by looking at its source code**
  - **Tests are generally structural**

White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed
- **Statement Coverage:** Every statement in the program is executed
- **Path Coverage:** Every possible path through the program is executed

Weaker

Stronger

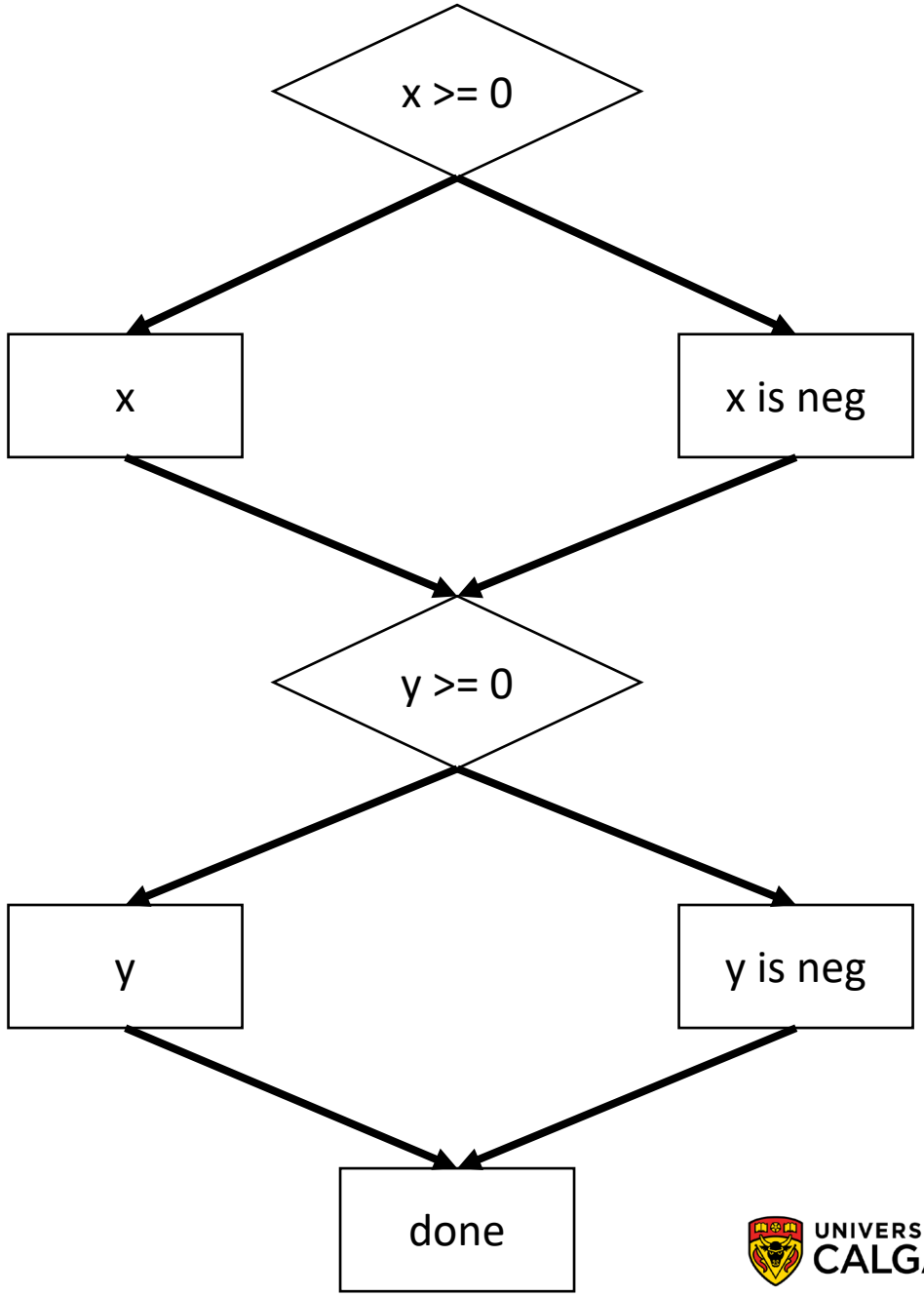
White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed
- **Statement Coverage:** Every statement in the program is executed
- **Path Coverage:** Every possible path through the program is executed

```
if x >= 0:
    print(x)
else:
    print(“x is neg”)
if y >= 0 :
    print(y)
else:
    print(“y is neg”)
print(“done”)
```

White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed
- **Statement Coverage:** Every statement in the program is executed
- **Path Coverage:** Every possible path through the program is executed



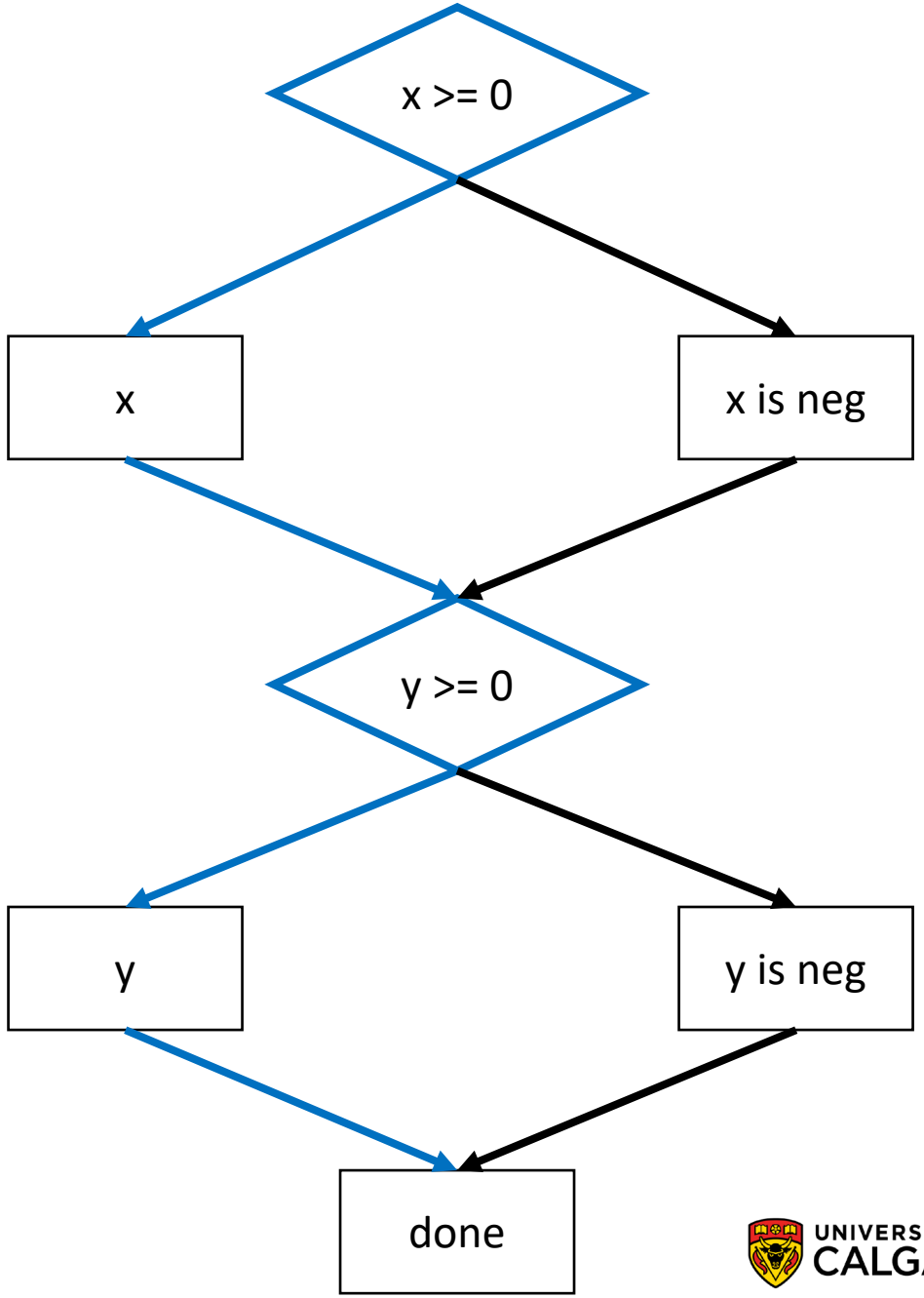
White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed

Test1: x=1, y=1

- **Statement Coverage:** Every statement in the program is executed

- **Path Coverage:** Every possible path through the program is executed



White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed

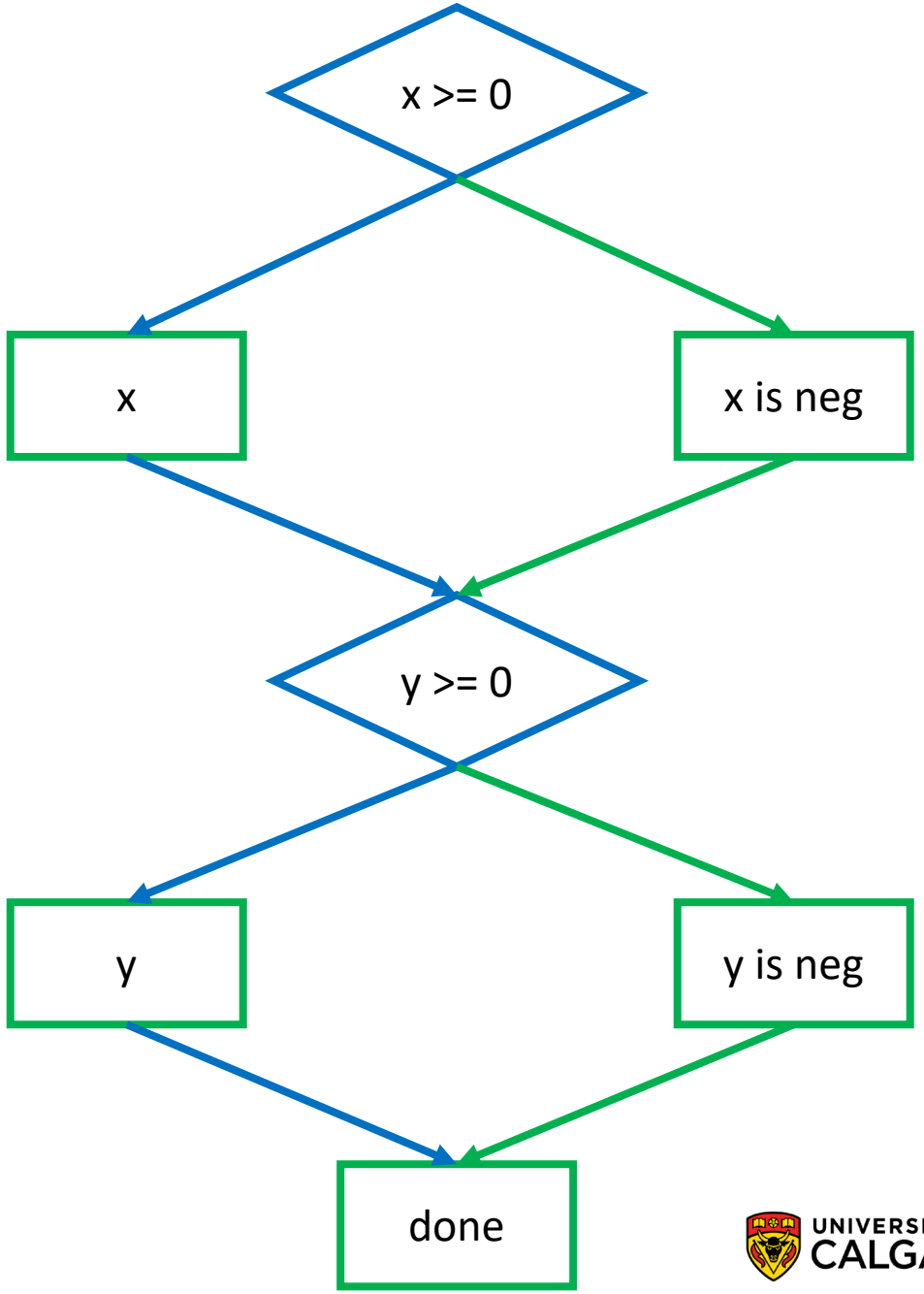
Test1: x=1, y=1

- **Statement Coverage:** Every statement in the program is executed

Test1: x=1, y=1

Test2: x=-1, y=-1

- **Path Coverage:** Every possible path through the program is executed



White Box Test Coverage

- **Condition Coverage:** Every decision point in the program is executed

Test1: x=1, y=1

- **Statement Coverage:** Every statement in the program is executed

Test1: x=1, y=1

Test2: x=-1, y=-1

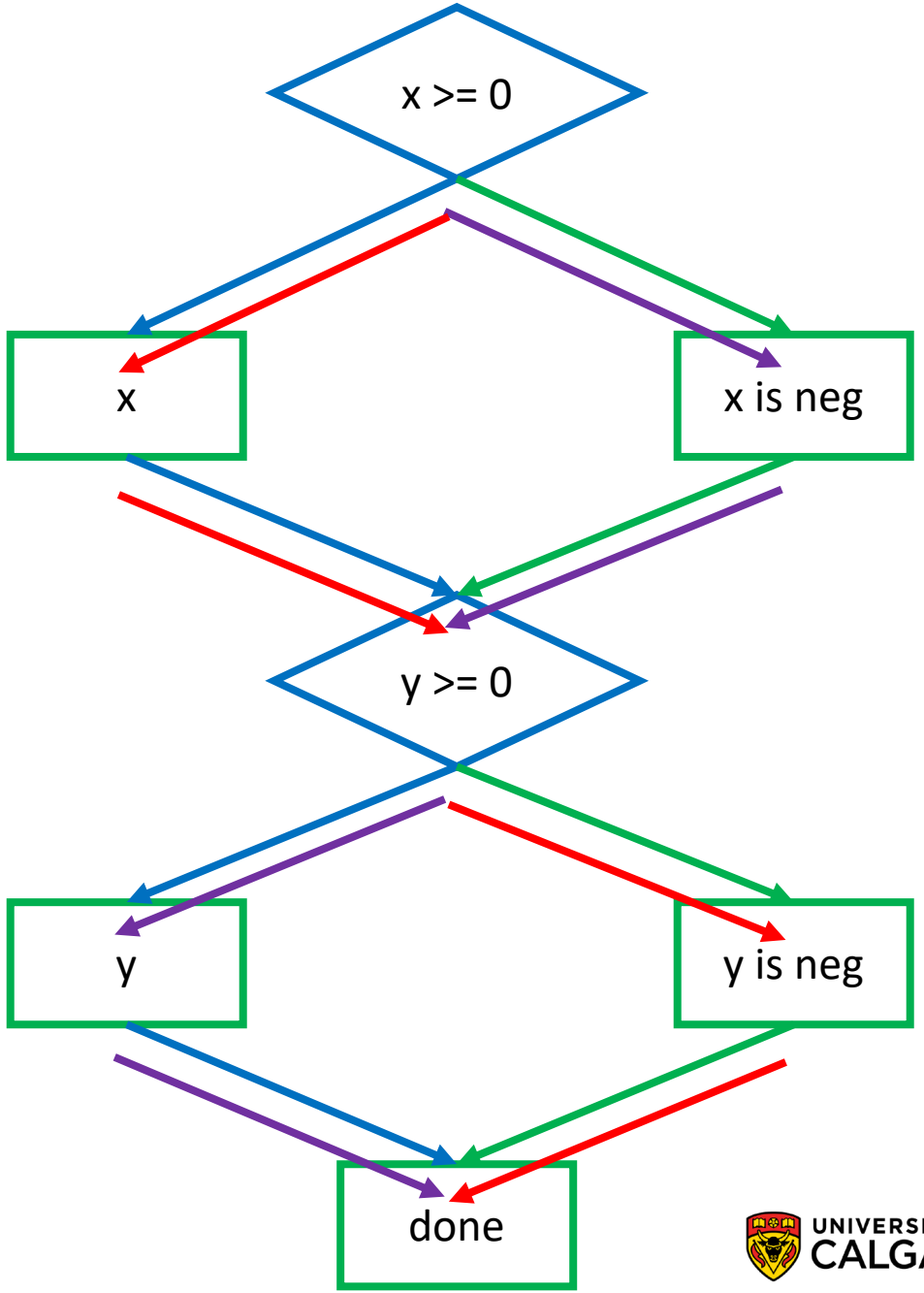
- **Path Coverage:** Every possible path through the program is executed

Test1: x=1, y=1

Test2: x=-1, y=-1

Test3: x=-1, y=1

Test4: x=1, y=-1



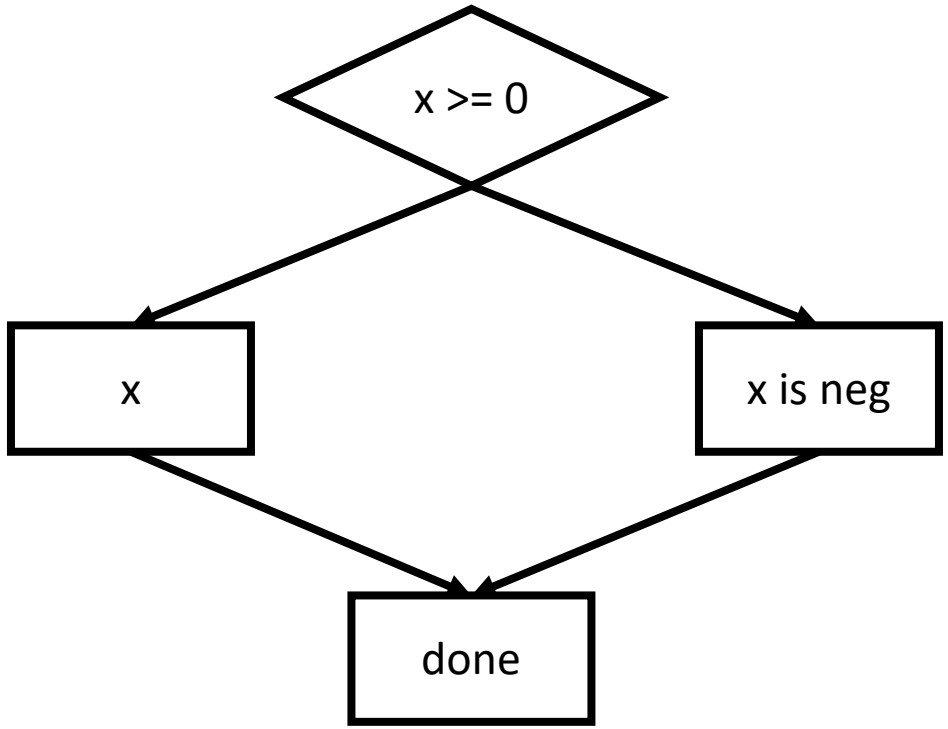
Boundary Testing

- A fourth type of white box testing
- Testing closely around border of decisions
- Improvement on path testing

```
if x >= 0:
   print(x)
else:
   print("x is neg")
```

Boundary is around 0

Test integers x=-1,0,1



Onward to … repetition.

Jonathan Hudson
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY