http://www.cs.cornell.edu/courses/cs1110/2022sp

Lecture 8:
Conditionals & Control Flow

(Sections 5.1-5.7)

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Lecture Afterthoughts

- Lots of written questions about Print vs Return
  - Please see: https://edstem.org/us/courses/19140/discussion/1084754
- A common post-lecture If-Elif-Else question:
  - https://edstem.org/us/courses/19140/discussion/1160274

Announcements

- **A1:** a1_first.py & policy_acknowledgement submission
  - Submit whatever you have at 2pm
  - Keep submitting as you make significant changes
  - Final submission due tonight at 11:59pm
- Conditionals—today's topic—not allowed in A1

What should I wear today?

```
def what_to_wear(temp):
    print("Today you should wear:")
    #  > 60: no jacket required
    # 40-60: jacket
    # 20-40: winter coat
    #  < 20: all the gear you own
```

How to we implement this in Python?

Conditionals: If-Statements

Format

```
if <boolean-expression>:
      <statement>
      …
      <statement>
```

Example

```
# is there a new high score?
if curr_score > high_score:
     high_score  = curr_score
     print("New high score!")
```

Execution:

if ⟨boolean-expression⟩ is true, then execute all of the statements indented directly underneath (until first non-indented statement)

What are Boolean expressions?

Expressions that evaluate to a Boolean value.

```
is_rainy = False
is_windy = True
temp = 12
```

Boolean variables:

```
if is_rainy:
   print(“Bring an umbrella!”)
```

Boolean operations:

```
if is_windy and not is_rainy:
   print(“Let's fly a kite!”)
```

Comparison operations:

```
if temp < 30 and is_rainy:
   print("Roads will be icy!")
```

```
if temp > 70:
     print("Hallelujah!")
```

What gets printed, Round 1

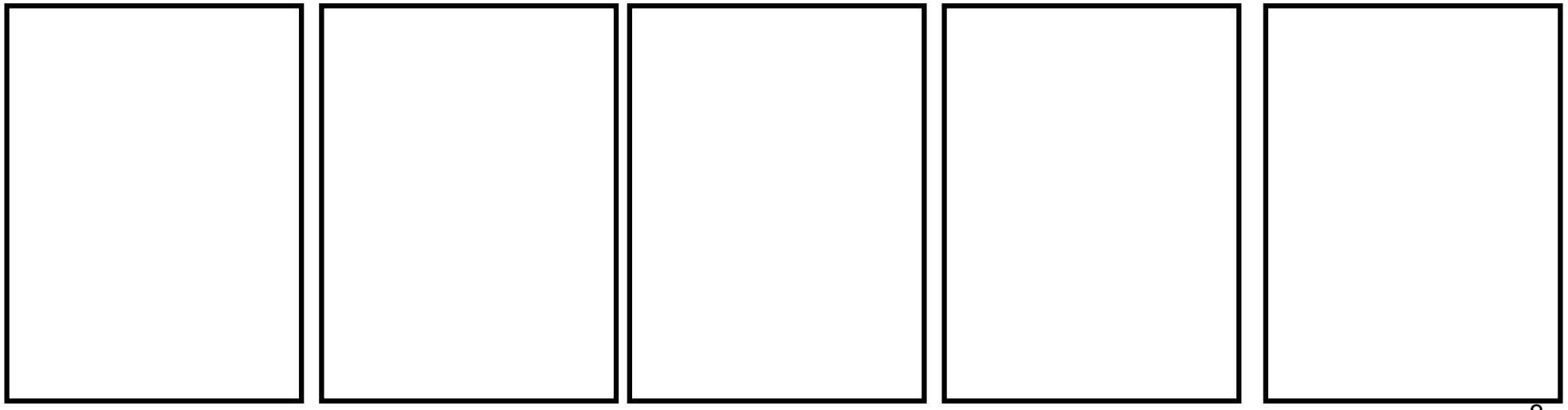
```
a = 0
print(a)
```

```
a = 0
a = a + 1
print(a)
```

```
a = 0
if a == 0:
    a = a + 1
print(a)
```

```
a = 0
if a == 1:
    a = a + 1
print(a)
```

```
a = 0
if a == 0:
    a = a + 1
a = a + 1
print(a)
```

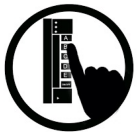


(Let's look at these one by one.)

What gets printed? (Question)

```
a = 0
if a == 0:
    a = a + 1
if a == 0:
    a = a + 2
a = a + 1

print(a)
```

A: 0
B: 1
C: 2
D: 3
E: I do not know



Conditionals: If-Else-Statements

Format

```
if <boolean-expression>:
    <statement>
    …
else:
    <statement>
    …
```

Example

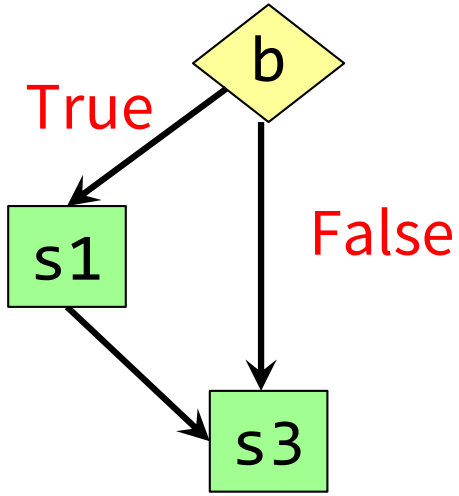
```
# new record?
if curr_score > high_score:
    print("New record!")
else:
    print("Nice try.")
```

Execution:

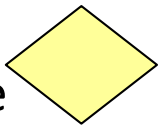
if ⟨boolean-expression⟩ is true, then execute statements indented under `if`; otherwise execute the statements indented under `else`

Conditionals: "Control Flow" Statements

```
if b:
    s1    # statement
s3        # statement
```

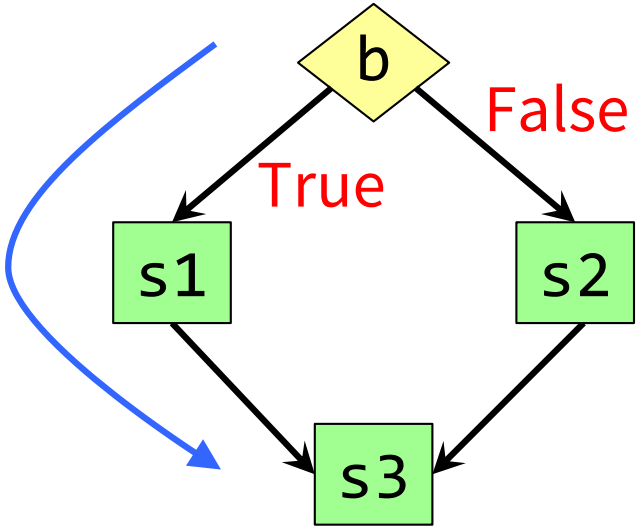


Branch Point:
Evaluate & Choose



Statements:
Execute

```
if b:
        s1
else:
        s2
s3
```



Flow

Program only takes one path during an execution (something will not be executed!)

What gets printed, Round 2

```
a = 0
if a == 0:
    a = a + 1
else:
    a = a + 2

print(a)
```

```
a = 0
if a == 1:
    a = a + 1
else:
    a = a + 2

print(a)
```

```
a = 0
if a == 1:
    a = a + 1
else:
    a = a + 2
a = a + 1
print(a)
```

```
a = 0
if a == 1:
    a = a + 1
else:
    a = a + 1
    a = a + 1
a = a + 1
print(a)
```

(Let's look at these one by one.)

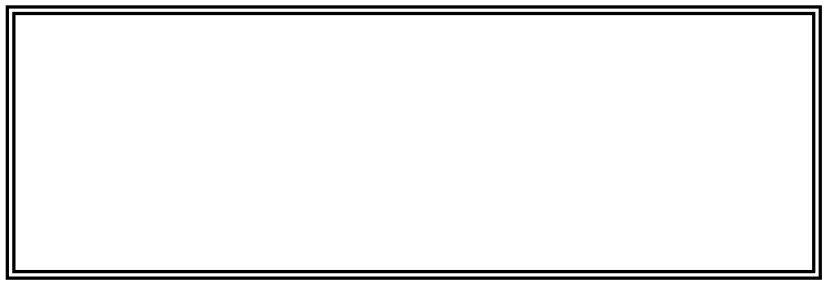



Program Flow (car locked, 0)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```

Global Space



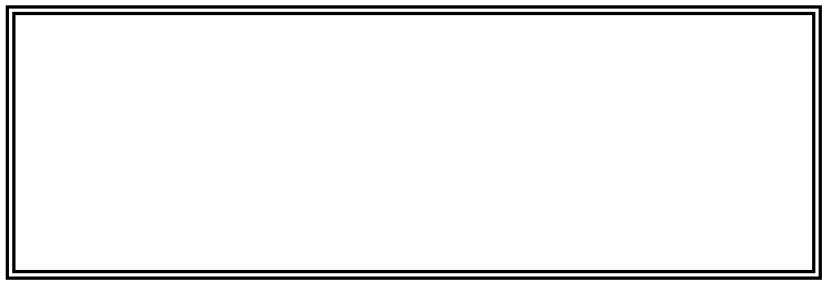
Program Flow (car locked, 1)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```

Global Space

`car_locked`	`True`



Program Flow (car locked, 2)

`if` determines which statement is executed next

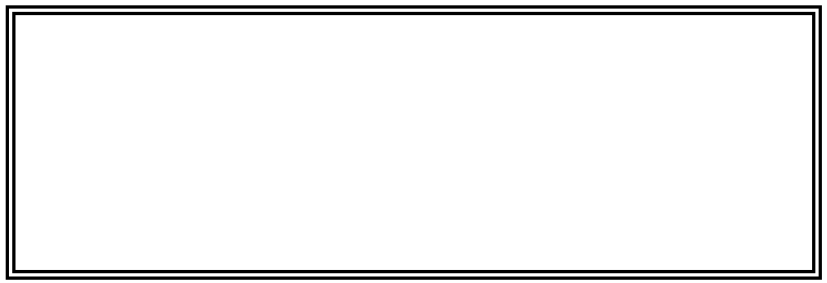
```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```

Global Space

car_locked	True

Call Stack

get_in_car	1
is_locked	True



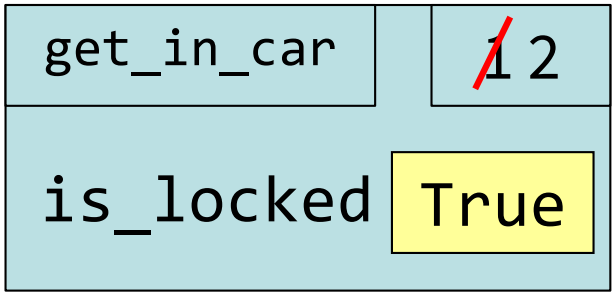
Program Flow (car locked, 3)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```

Global Space

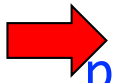
Call Stack



Program Flow (car locked, 4)

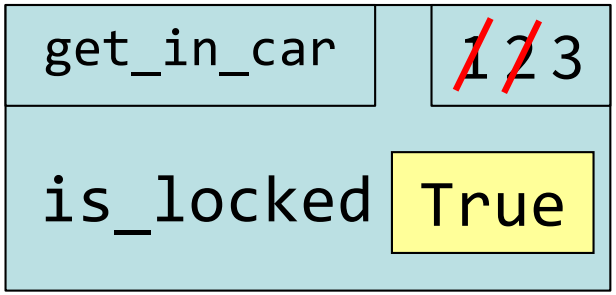
`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```



Global Space

Call Stack



Unlock car!

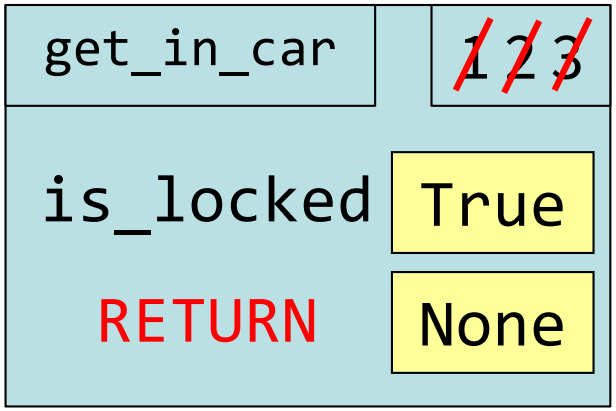
Program Flow (car locked, 5)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```

Global Space

Call Stack

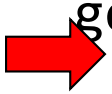



```
Unlock car!
Open the door.
```

Program Flow (car locked, 6)

`if` determines which statement is executed next

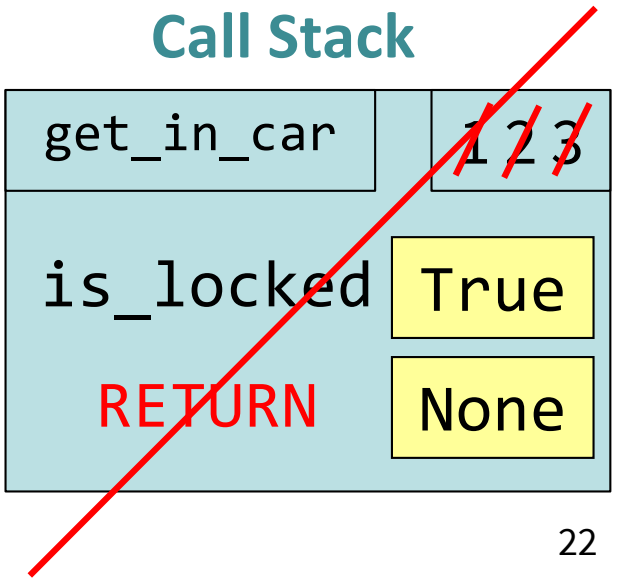
```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = True
   get_in_car(car_locked)
```



Global Space

car_locked	True

Call Stack



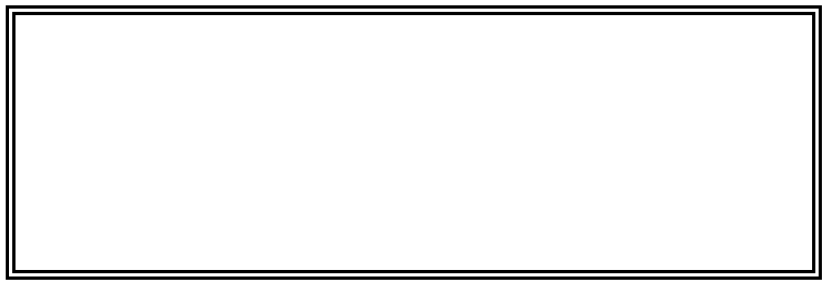
Unlock car!
Open the door.

Program Flow (car not locked, 0)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```

Global Space



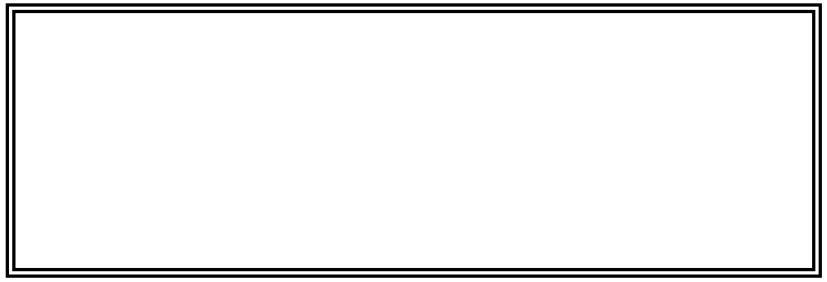
Program Flow (car not locked, 1)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```

Global Space

car_locked	False



Program Flow (car not locked, 2)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```

Global Space

car_locked	False

Call Stack

get_in_car	1
is_locked	False

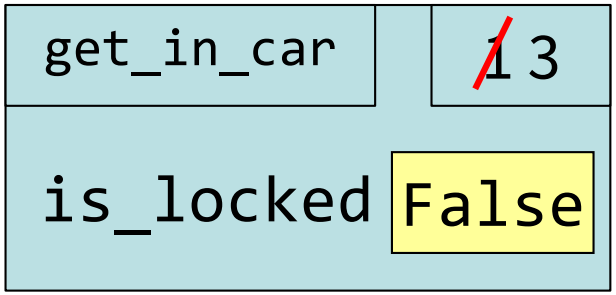
Program Flow (car not locked, 3)

`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```

Global Space

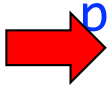
Call Stack



Program Flow (car not locked, 4)

`if` determines which statement is executed next

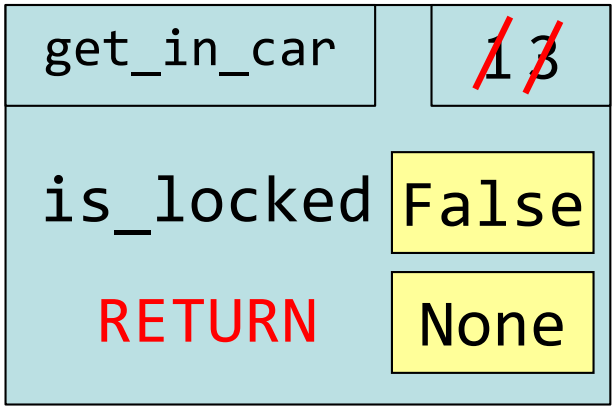
```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```



Global Space

car_locked	False

Call Stack



Open the door.

Program Flow (car not locked, 5)

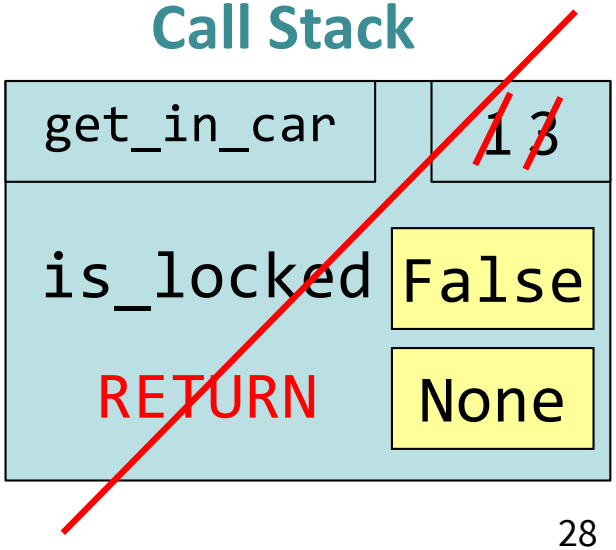
`if` determines which statement is executed next

```
   def get_in_car(is_locked):
1      if is_locked:
2          print("Unlock car!")
3      print("Open the door.")

   car_locked = False
   get_in_car(car_locked)
```

Global Space

car_locked	False

Call Stack

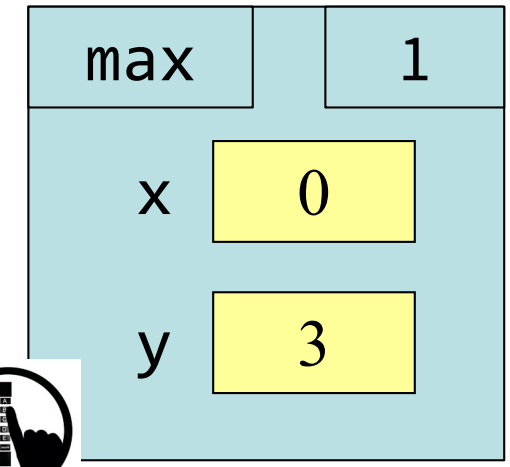


Open the door.

What does the call frame look like next? (Q)

```
   def max(x,y):
1      if x > y:
2          return x
3   return y

   max(0,3)
```

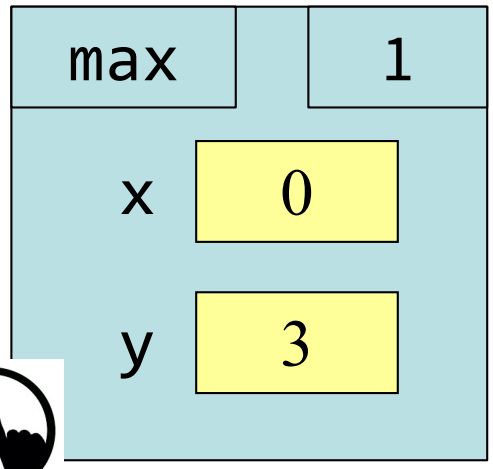
Current call frame:



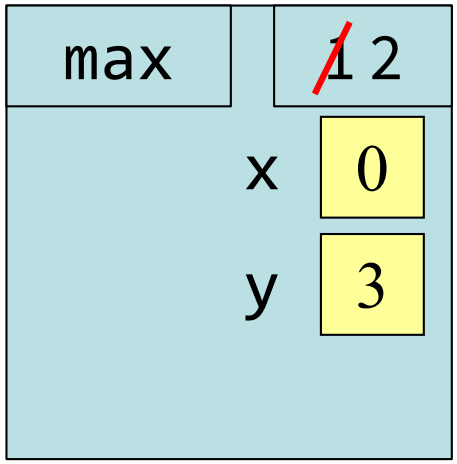
What does the call frame look like next? (Q)

```
def max(x,y):
    if x > y:
1       return x
2   return y
3
max(0,3)
```

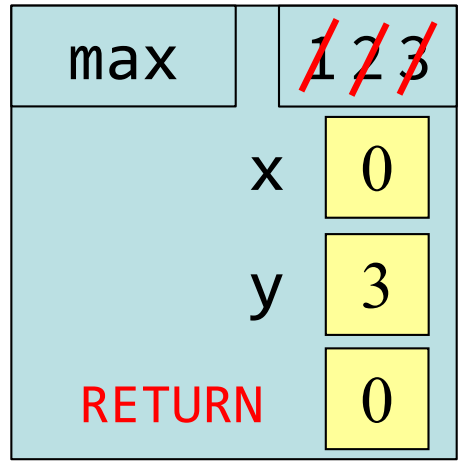
Current call frame:



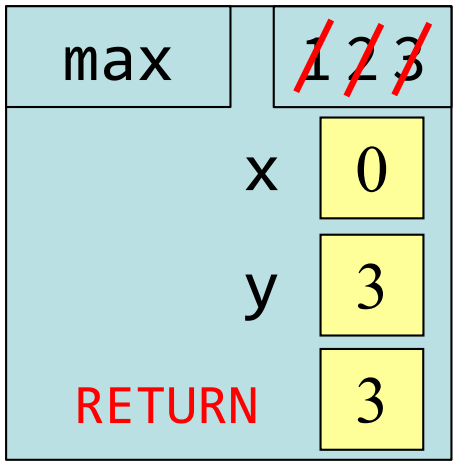
A:



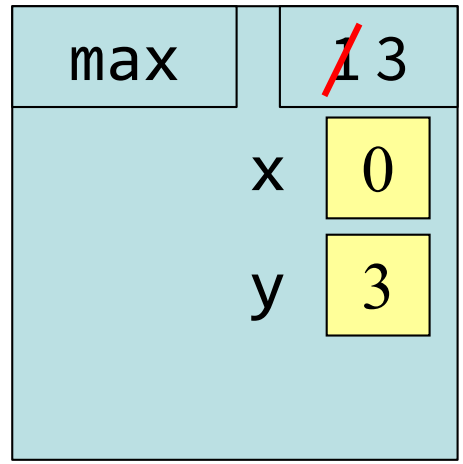
B:



C:



D:



Program Flow and Variables

Variables created inside `if` continue to exist past `if`:

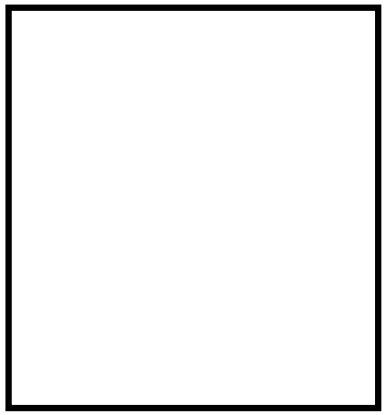
```
a = 0
if a == 0:
    b = a + 1
print(b)
```

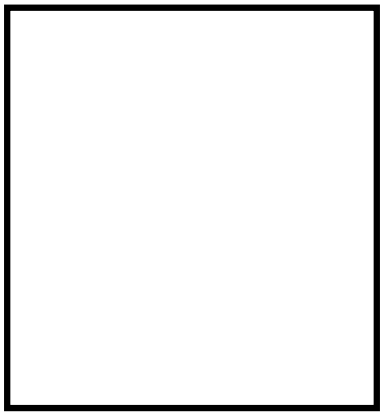
…but are only created if the program actually executes that line of code

What gets printed, Round 3

```
a = 0
if a == 0:
    b = 0
print(b)
```

```
a = 1
if a == 0:
    b = 0
print(b)
```



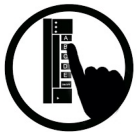


Control Flow and Variables (Q1)

```
def max(x,y):
    """Returns: max of x, y"""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(3,0)
```

Value of `maximum`?

A: 3
B: 0
C: Error!
D: I do not know

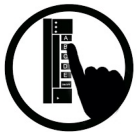


Control Flow and Variables (Q2)

```
def max(x,y):
    """Returns: max of x, y"""
    # note: code has a bug!
    # check if x is larger
    if x > y:
        bigger = x
    return bigger

maximum = max(0,3)
```

Value of `maximum`?

A: 3
B: 0
C: Error!
D: I do not know



Program Flow and Variables

```
def zero_or_one(a):
    if a == 1:
        b = 1
    else:
        b = 0
    print(b)
```

make sure that ALL `if` branches create the variable

Conditionals: If-Elif-Else-Statements (1)

Format

```
if <Boolean expression>:
    <statement>
    …
elif <Boolean expression>:
    <statement>
    …
    …
else:
    <statement>
    …
```

Example

```
# Find the winner
if score1 > score2:
    winner  = “Player 1”
elif score2 > score1:
    winner  = “Player 2”
else:
    winner = “Players 1 and 2"
```

Conditionals: If-Elif-Else-Statements (2)

Format

```
if <Boolean expression>:
    <statement>
    …
elif <Boolean expression>:
    <statement>
    …
    …
else:
    <statement>
    …
```

Notes on Use

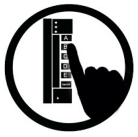
- No limit on number of `elif`
  - Must be between `if`, `else`
- `else` is optional
  - `if-elif` by itself is fine
- Booleans checked in order
  - Once Python finds a true <Boolean-expression>, skips over all the others
  - `else` means **all** <Boolean-expression> are false

If-Elif-Else (Question)

```
a = 2

if a == 2:
    a = 3
elif a == 3:
    a = 4
print(a)
```

What gets printed?

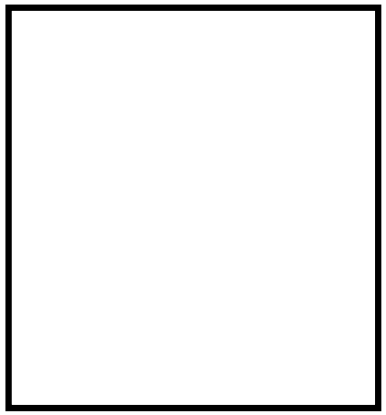
A: 2
B: 3
C: 4
D: I do not know

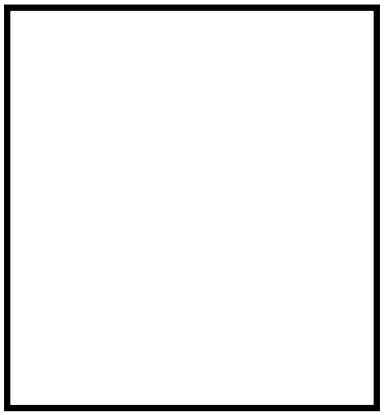


What gets printed, Round 4

```
a = 2
if a == 2:
    a = 3
elif a == 3:
    a = 4
print(a)
```

```
a = 2
if a == 2:
    a = 3
if a == 3:
    a = 4
print(a)
```





The logic can get a little dizzying…

```
def what_to_wear(raining, freezing):
    if raining and freezing:
        print(”Wear a waterproof coat.”)
    elif raining and not freezing:
        print(”Bring an umbrella.")
    elif not raining and freezing:
        print(”Wear a warm coat!")
    else:
        print(”A sweater will suffice.")
```

Nested Conditionals to the rescue!

```
def what_to_wear(raining, freezing):
    if raining:
        if freezing:
            print(”Wear a waterproof coat.”)
        else:
            print(”Bring an umbrella.")
    else:
        if freezing:
            print(”Wear a warm coat!")
        else:
            print(”A sweater will suffice.")
```

Program Flow and Testing

```
# determine winner


if x_score > y_score:
     winner = "x"
else:
     winner = "y"
```

Can use `print` statements to examine program flow

Program Flow and Testing

```
# determine winner
print('before the if’)
if x_score > y_score:
    print(‘inside the if’)
    winner = "x"
else:
    print(‘inside the else’)
    winner = "y"
print('after the if’)
```

Can use `print` statements to examine program flow

“traces” or “breadcrumbs”

x_score must have been greater than y_score

```
'before the if'
'inside the if'
'after the if'
```

Traces (control) and Watches (data)

```
# determine winner
print('before the if’)
if x_score > y_score:
    print(‘inside the if’)
    winner = "x"
    print('winner = '+winner)
else:
    print(‘inside the else’)
    winner = "y"
    print('winner = '+winner)
print('after the if’)
```

← TRACES

Trace **program flow**

What code is being executed? Place print statements at the beginning of a code block that might be skipped.

← WATCHES

Watch **data** values

What is the value of a variable? Place print statements after assignment statements.