Hack110!!!

When: Saturday, April 18th from 10 am - 12 am (MIDNIGHT)

Where: Sitterson and Fred Brooks 1st floor (Floor 0)

Who can join? Anyone in COMP 110! No prior experience required.

Perks :

- Snacks and drinks All Day
- Free Shirt and CLE Credit
- Lunch and Dinner provided
- Fun workshops
- Demos for Prizes

Announcements

Re: Assignments:

- **LS07: Variables** due tonight at 11:59pm
- **EX02: Chardle** – due Sunday, Feb 8 at 11:59pm

Reminders:

- **Quiz 01 on Thurs, Feb 12**
  - If you take your quizzes with the EOC/ARS, please ensure you've scheduled it!
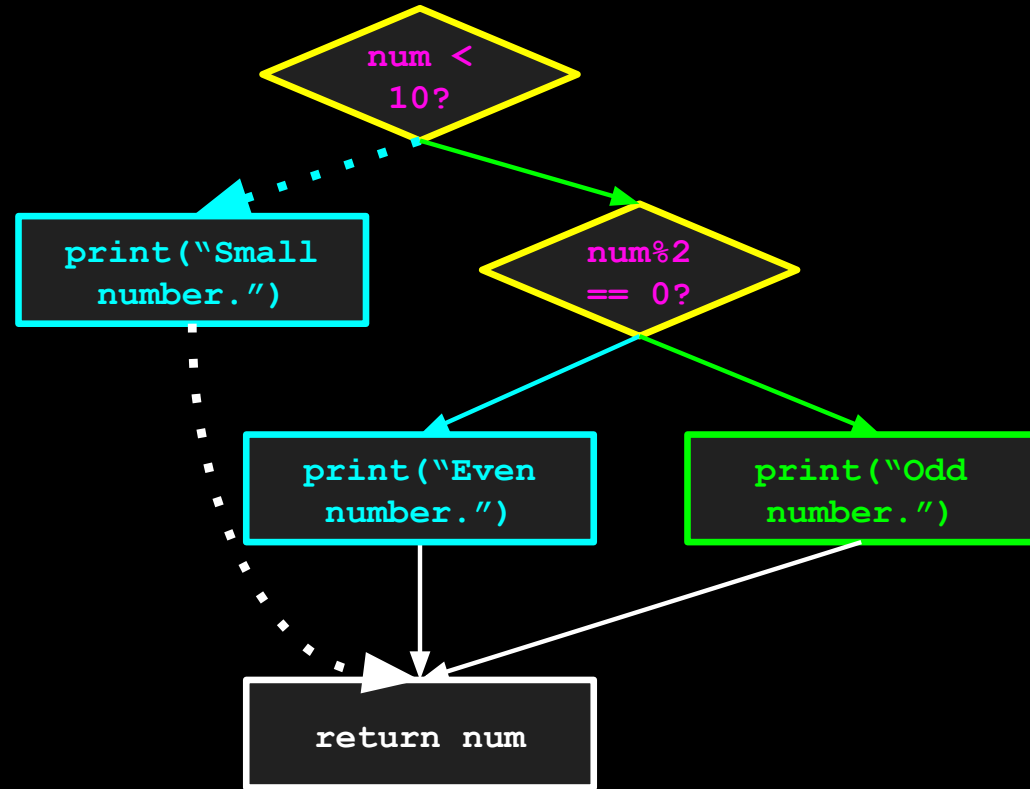
COMP 110

Conditionals (continued)

Drawing the control flow…

```
1    def number_info(num: int) -> None:
2        if num < 10:
3            print("Small number.")
4        else:
5            if num % 2 == 0:
6                print("Even number.")
7            else:
8                print("Odd number.")
9        return num
10
11   number_info(num=11)
12   print(number_info(num=4))
```
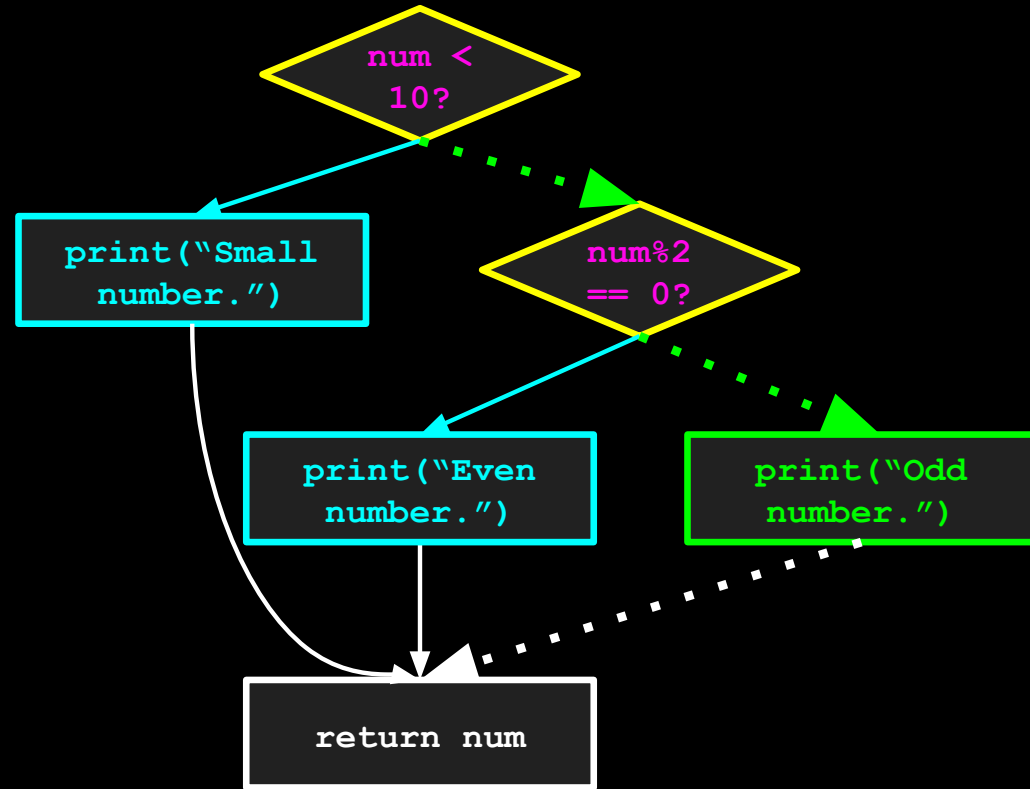
Drawing the control flow…

```
1    def number_info(num: int) -> None:
2        if num < 10:
3            print("Small number.")
4        else:
5            if num % 2 == 0:
6                print("Even number.")
7            else:
8                print("Odd number.")
9        return num
10
11   number_info(num=11)
12   print(number_info(num=4))
```
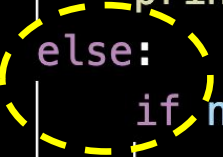
What if…

```
1   def number_info(num: int) -> None:
2       if num < 10:
3           print("Small number.")
4       else:
5           if num % 2 == 0:
6               print("Even number.")
7           else:
8               print("Odd number.")
9       return num
```

What if…

```
1    def number_info(num: int) -> None:
2        if num < 10:
3            print("Small number.")
4        else:
5            if num % 2 == 0:
6                print("Even number.")
7            else:
8                print("Odd number.")
9        return num
```
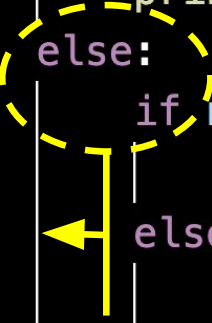
```
1    def number_info(num: int) -> None:
2        if num < 10:
3            print("Small number.")
4        elif num % 2 == 0:
5            print("Even number.")
6        else:
7            print("Odd number.")
8        return num
```

What if…

```
1  def number_info(num: int) -> None:
2      if num < 10:
3          print("Small number.")
4      else:          elif
5          if num % 2 == 0:
6              print("Even number.")
7          else:
8              print("Odd number.")
9      return num
```

```
1  def number_info(num: int) -> None:
2      if num < 10:
3          print("Small number.")
4      elif num % 2 == 0:
5          print("Even number.")
6      else:
7          print("Odd number.")
8      return num
```

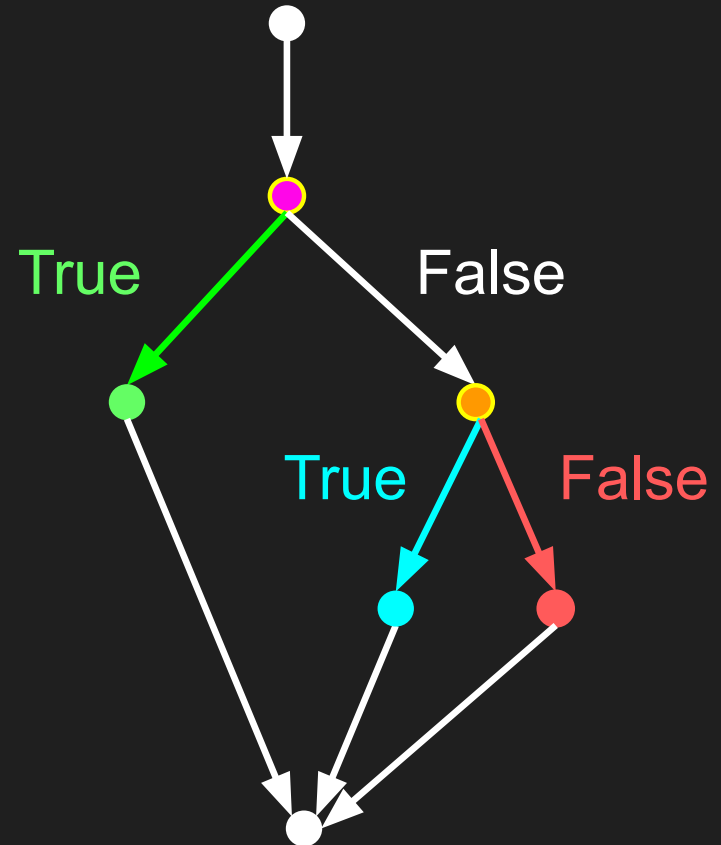# Previous Control Flow

if <condition>:

    <do something>

else:

    if <other condition>:

        <do something else>

    else:

        <do third thing>

<rest of program>

New Control Flow

if <condition>:

    <do something>

elif <other condition>:

    <do something else>

else:

    <do third thing>

<rest of program>

<condition>
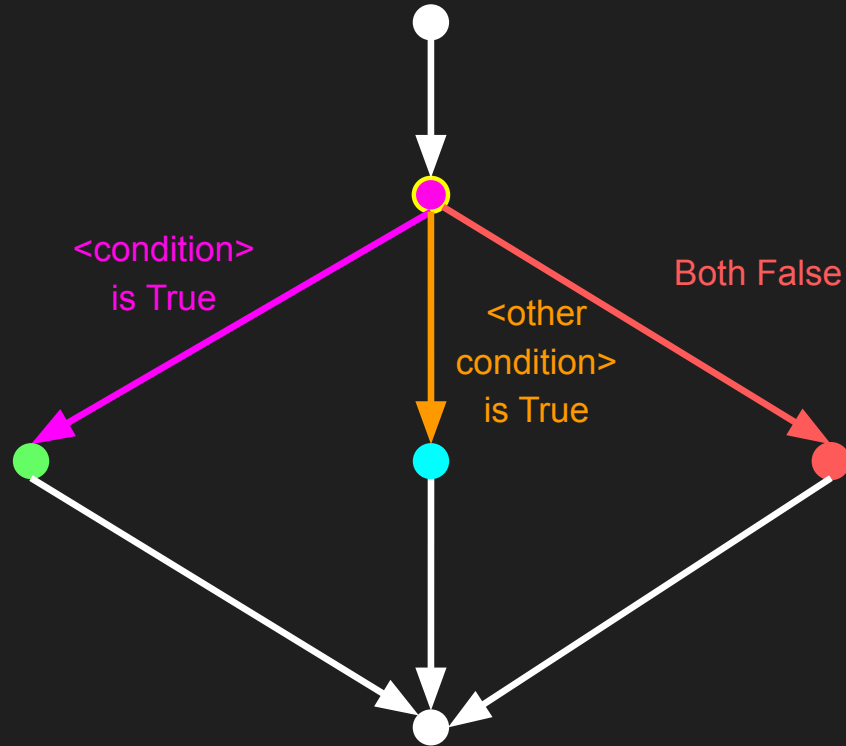is True

<other
condition>
is True

Both False

COMP 110

Variables &
Positional Arguments

Warm-Up: Discuss these questions with a neighbor, then diagram how you believe this works:

Note: This warm-up gives a preview of the new concepts we'll cover today. As you work through it, focus on using your intuition and what you already know to make sense of the code.

```
1    def f(x: int) -> int:
2        y: int
3        y = x * 2
4        return y
5
6
7    print(f(3))
```

Questions to discuss with a neighbor:
What does line 2 remind you of?
What does line 3 remind you of?

```
1    def f(x: int) -> int:
2        y: int
3        y = x * 2
4        return y
5
6
7    print(f(3))
```

Output

Stack

Globals

Heap

```
1    def f(x: int) -> int:
2        y: int
3        y = x * 2
4        return y
5
6
7    print(f(3))
```

Output

6

Stack

Globals

f | id: 0

f

RA | 7   x | 3
RV | 6   y | 6

Heap

id: 0 | fn lines 1-4

Key Variable Terminology

Variable Declaration / Definition   -   Associates a name/identifier with a data type, and a space in the current frame

`<name>: <type>`

Examples:

`students: int`

`message: str`

Key Variable Terminology

Variable Declaration / Definition

`<name>`: `<type>`

- Associates a name/identifier with a data type, and a space in the current frame

Variable Assignment

`<name>` = `<expression>`

Examples:

`students = 900`

`message = "comp" + str(110)`

- Binds a new value to a variable name in memory

Key Variable Terminology

Variable Declaration / Definition   -   Associates a name/identifier with a
                                           data type, and a space in the current
`<name>: <type>`   frame

Variable Assignment   -   Binds a new value to a variable name
                                           in memory
`<name> = <expression>`

Variable Initialization   -   First time a variable is assigned

Key Variable Terminology

Variable Declaration / Definition - Associates a name/identifier with a data type, and a space in the current frame

`<name>: <type>`

Variable Assignment - Binds a new value to a variable name in memory

`<name> = <expression>`

Variable Initialization - First time a variable is assigned

Note: You can declare and initialize a variable on two different lines (e.g., lines 2-3 in the warm-up), or on the same line, e.g:
```
y: int = x * 2
```

Key Variable Terminology

```python
1   def f(x: int) -> int:
2       y: int
3       y = x * 2
4       return y
5
6
7   print(f(3))
```

Variable Initialization   -   First time a variable is assigned

Note: You can declare and initialize a variable on two different lines (e.g., lines 2-3 in the warm-up), or on the same line, e.g:
`y: int = x * 2`

Key Variable Terminology

Variable Declaration / Definition   - Associates a name/identifier with a data type, and a space in the current frame

`<name>: <type>`

Variable Assignment   - Binds a new value to a variable name in memory

`<name> = <expression>`

Variable Initialization   - First time a variable is assigned

Variable Access   - "Reading" or using a variable name in an expression

`print(students)`

Left-hand vs. Right-hand Side of Assignment

Each side of the assignment operator (=) plays a distinct role in variable assignment!

Variable Assignment

`<name> = <expression>`

Execute an assignment statement in 2 steps:

1. Evaluate the **expression** on the right-hand side of the **assignment operator**, down to a literal value.
2. Assign (or "bind") that literal value to the **variable** on the left-hand side.

Common Variable Errors

UnboundLocalError – Occurs when attempting to access a variable that is declared in a function but not yet initialized

NameError – Occurs when attempting to access a variable that has not been declared. Commonly from typos or renaming a variable and not updating all accesses

Why variables?   One reason: to store the results of function calls for later use!

```python
def pizza_price(size: int, toppings: int) -> float:
    """Calculate price of pizza with toppings."""
    price: float = 10.0

    if size >= 16:
        price = 20.0

    price = price + toppings * 0.75

    return price


total_price: float = pizza_price(size=14, toppings=2)
print(total_price)
```

```python
1   def pizza_price(size: int, toppings: int) -> float:
2       """Calculate price of pizza with toppings."""
3       price: float = 10.0
4
5       if size >= 16:
6           price = 20.0
7
8       price = price + toppings * 0.75
9
10      return price
11
12
13    total_price: float = pizza_price(size=14, toppings=2)
14    print(total_price)
```
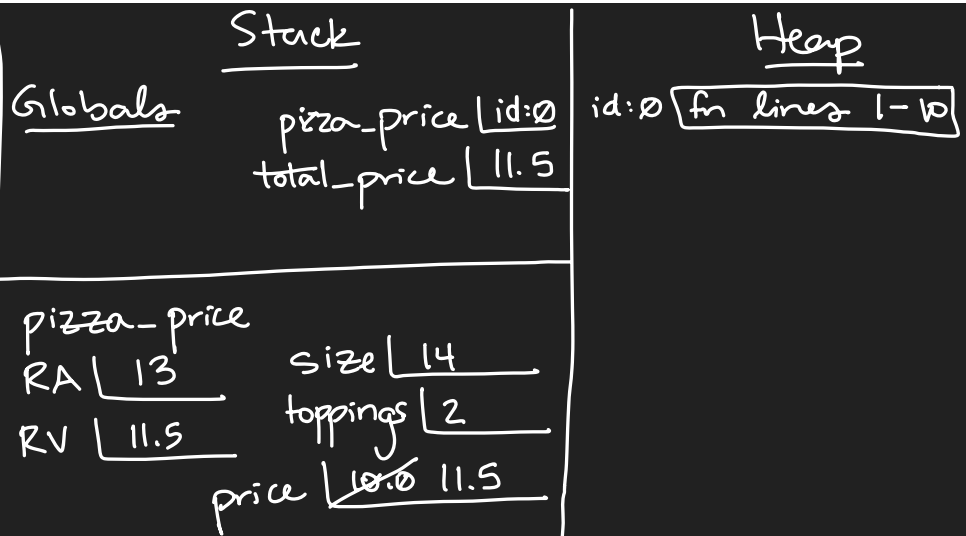
Line 8 annotation:
10.0 + 2 * 0.75
1.5

Stack

Globals
pizza_price [id:0]
total_price [11.5]

pizza_price
RA [13]
RV [11.5]
size [14]
toppings [2]
price [10.0 11.5]

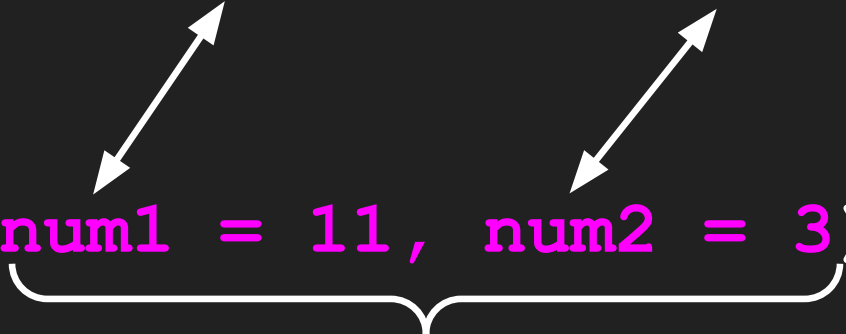## Heap

id:0 [fn lines 1-10]

Output

11.5

Keyword vs. Positional Arguments

Recall: Signature vs Call

```python
def divide(num1: int, num2: int) -> float:



divide(num1 = 11, num2 = 3)
```

These are called keyword arguments, since you are assigning values based on the parameter names.

Keyword arguments

```python
def divide(num1: int, num2: int) -> float:



divide(num1 = 11, num2 = 3)
```

Benefit of keyword arguments: order of arguments doesn't matter.

Keyword arguments

```python
def divide(num1: int, num2: int) -> float:



divide(num1 = 11, num2 = 3)

divide(num2 = 3, num1 = 11)
```

Benefit of keyword arguments: order of arguments doesn't matter.

Positional Arguments

```python
def divide(num1: int, num2: int) -> float:



divide(11, 3)
```

For positional arguments, values are assigned based on the order (*position*) of the arguments.

```python
def pizza_price(size: int, toppings: int) -> float:
    """Calculate price of pizza with toppings."""
    price: float = 10.0

    if size >= 16:
        price = 20.0

    price = price + toppings * 0.75

    return price


total_price: float = pizza_price(size=14, toppings=2)
print(total_price)
```

How could we rewrite these as **positional arguments**?

pizza_price (14, 2)

```python
"""Calling to and fro..."""


def ping(i: int) -> int:
    print("ping: " + str(i))
    if i <= 0:
        return i
    else:
        return pong(i=i - 1)


def pong(i: int) -> int:
    print("pong: " + str(i))
    return ping(i=i - 1)


print(ping(i=2))
```

```
 1    """Calling to and fro..."""
 2
 3
 4    def ping(i: int) -> int:
 5        print("ping: " + str(i))
 6        if i <= 0:
 7            return i
 8        else:
 9            return pong(i=i - 1)
10
11
12    def pong(i: int) -> int:
13        print("pong: " + str(i))
14        return ping(i=i - 1)
15
16
17    print(ping(i=2))
```

ping(2)

How could we rewrite this as a **positional argument**?

Weekly Tutoring + Office Hours

Office Hours (Sitterson Hall (SN) 008):

- Mondays–Fridays: 11am-5pm
- Sundays: 1-5pm

Tutoring (see CSXL site for location):

- Mondays, Wednesdays, Thursdays: 5-7pm