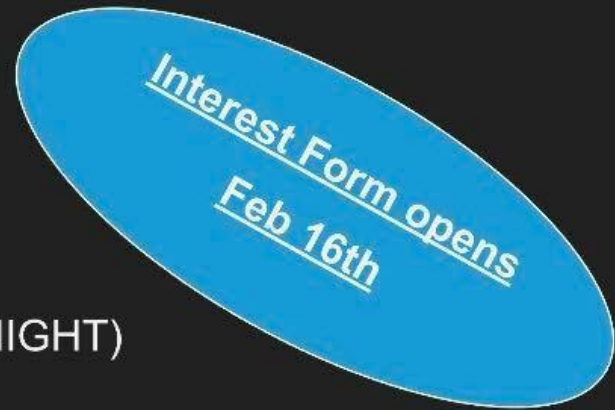




# Hack110!!!



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





CL10: Variables, Positional  
Arguments, & Mem. Diag. Practice

# Announcements

## Re: Assignments:

- **LS07: Variables** due tonight at 11:59pm
  - [Read this reading](#) about variables!
- **EX02: Chardle** – due Sunday, Feb 8 at 11:59pm

## Reminders:

- **Quiz 00 regrade requests** open till Friday at 11:59pm
- **Quiz 01 on Friday, Feb 13**
  - If you take your quizzes with the EOC/ARS, please ensure you've scheduled it!
  - If you will have a university-approved absence for this day, tell me ASAP so we can arrange a time for you to make it up!

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

```
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))
```

# 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

`students = 300`

- Binds a new value to a variable name in memory

# 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

`students = 300`

- Binds a new value to a variable name in memory

Variable Initialization

- First time a variable is assigned

# 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

`students = 300`

- Binds a new value to a variable name in memory

Variable Initialization

- First time a variable is assigned

Variable Access

- “Reading” or using a variable name in an expression

# Left-hand vs. Right-hand Side of Assignment

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

# 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!

```
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)
```

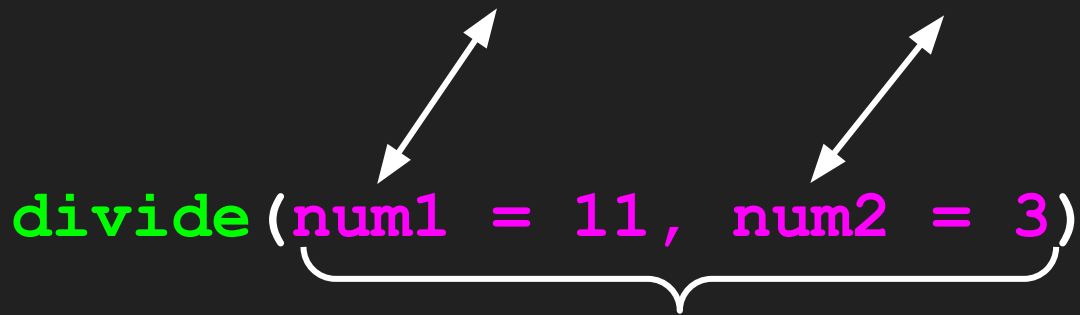
```
1 def pizza_price(size: int, toppings: int) -> float:
2     """Calculate price of pizza with toppings."""
3     price: float = 10.0
4
5     if size >= 16:
6         price = 20.0
7
8     price = price + toppings * 0.75
9
10    return price
11
12
13 total_price: float = pizza_price(size=14, toppings=2)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

# Keyword vs. Positional Arguments

## Recall: Signature vs Call

```
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

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

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



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

# Keyword arguments

```
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


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

```
divide(11, 3)
```

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

```
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)
```

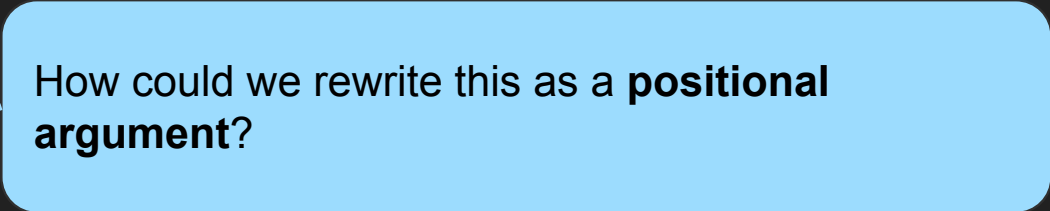
How could we rewrite these as **positional arguments**?



```
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))
```

```
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))
```

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