



CS 1112: Introduction To Programming

`def functions`(An Introduction)

Dr. Nada Basit // `basit[at]Virginia[dot]edu`

Spring 2024

Friendly Reminders

- Your **safety** and **comfort** is important!
 - If you choose to wear a mask you are welcome to do so
 - *We will interpret wearing a mask as being considerate and caring of others in the classroom (not that you are sick), and realize that some may choose to mask to remain distanced*
- Be an **active** participant in your learning!
You're welcome and **encouraged** to ask questions during class!
- If you feel **unwell**, or think you are, **please stay home**
 - *We will work with you!*
 - Get some rest 😊
 - View the recorded lectures – *please allow 24-48 hours to post*
 - *Contact us!*



Announcements

- **Quiz 3** is **due by 11:00pm on Monday (tonight)**!
 - No late quizzes accepted
 - No make-up quizzes allowed
 - If you believe your computer is glitching, it's a good idea to copy down your answers to each of the questions in a word document. In the event something happens, you can send me your solutions.
 - ***Note:** in general, will **cannot and will not** accept quiz solutions via **email**. We will only accept them in the case where your quiz may have glitched and we **no longer have your submitted answers**.*
 - **Take quiz on:** [Sherlock.cs.virginia.edu](https://sherlock.cs.virginia.edu)
- **PA01** is graded! Check scores on Gradescope and on Canvas
- **PA02** is out and is **due by 11:00pm on Wednesday (9/20)**!
 - Submit on Gradescope: your .py file and a reflection file (PDF).
 - *Not sure how to create/submit a PDF? No problem! Ask one of the TAs for help!*

Functions ... *Like getting takeout from Bodos*

- Enter with your order in mind and your debit card
 - This is the **input** to the function. We call this passing “**arguments**” to the function
- Give the restaurant your order and your debit card, which will be turned in to the actual food
- The restaurant records your order and your debit card info
 - The arguments that you passed in are **stored by the function**. The variables in the function that store the arguments are call “parameters”.
- People create the food items you wanted
 - This could include **many, many steps** (getting and preparing each ingredient, cooking, etc.)
- Things happen within the function that **impact the world outside** of the function
 - This is known as a “**side effect**”
 - A side effect of this function is that you **have less money in you bank account**
- Leave with your food
 - **Return** value: the **food**

Reasons to Use Functions

???

Reasons to Use Functions

- 1. reduce complex tasks into **simpler tasks**
- 2. **eliminate duplicate code** (no need to re-write, reuse function as needed)
- 3. **code reuse** (once function is written, can **reuse** it in any other program)
- 4. **distribute tasks** to multiple programmers (each function written by someone)
- 5. **hide implementation details** - **abstraction** (increase readability, increase maintenance and quality)
- 6. **improves debugging** by improving **traceability** (easier to follow - jump from function to function)

Creating a Function

def name(parameters):

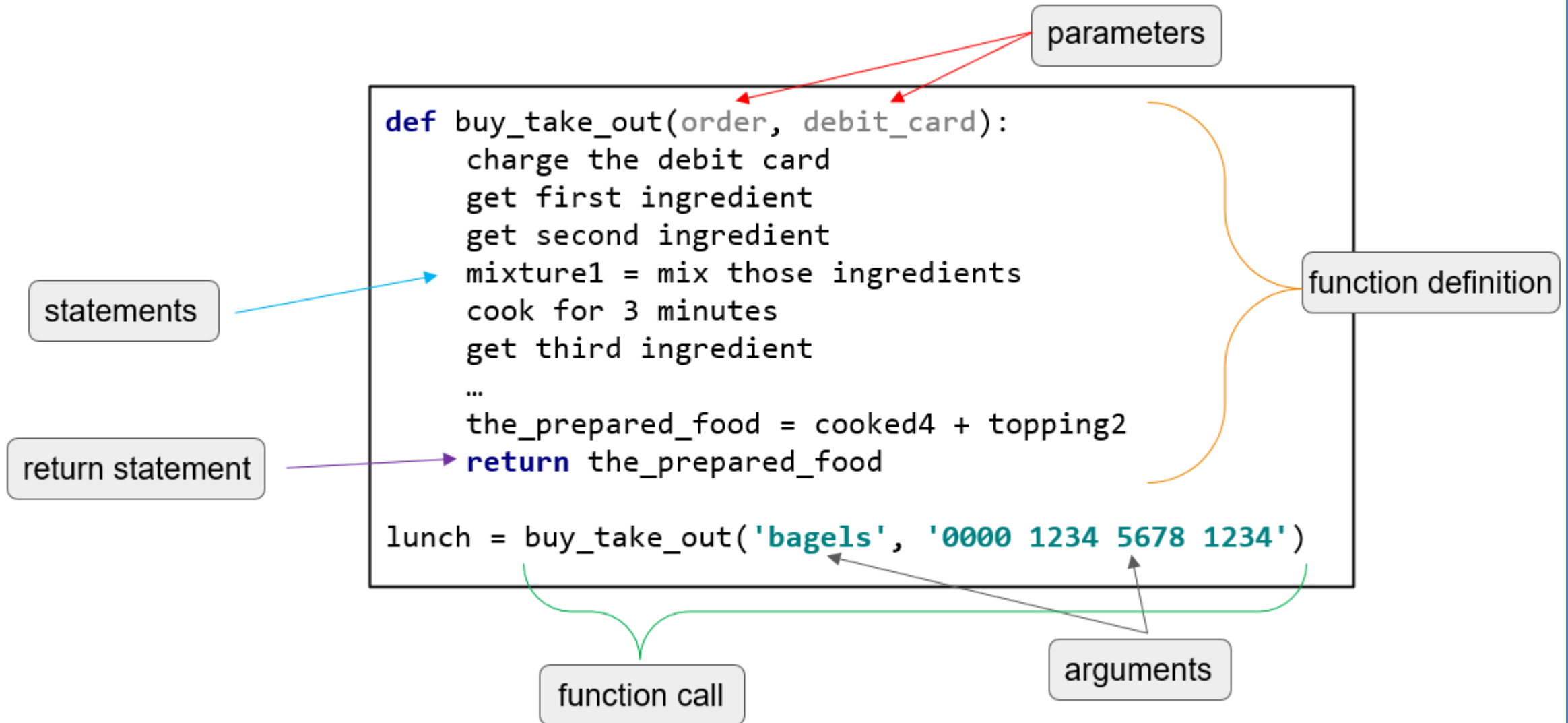
indent

Statements...
(Do these things)

return something

```
def buy_take_out(order, debit_card):  
    charge the debit card  
    get first ingredient  
    get second ingredient  
    mixture1 = mix those ingredients  
    cook for 3 minutes  
    get third ingredient  
    ...  
    the_prepared_food = cooked4 + topping2  
    return(the_prepared_food)  
  
buy_take_out('bagels', '0000 1234 5678 1234')
```

Parts of a Function




```
def buy_take_out(order, debit_card):  
    charge the debit card  
    get first ingredient  
    get second ingredient  
    mixture1 = mix those ingredients  
    cook for 3 minutes  
    get third ingredient  
    ...  
    the_prepared_food = cooked4 + topping2  
    return(the_prepared_food)  
  
lunch = buy_take_out('bagels', '0000 1234 5678 1234')
```

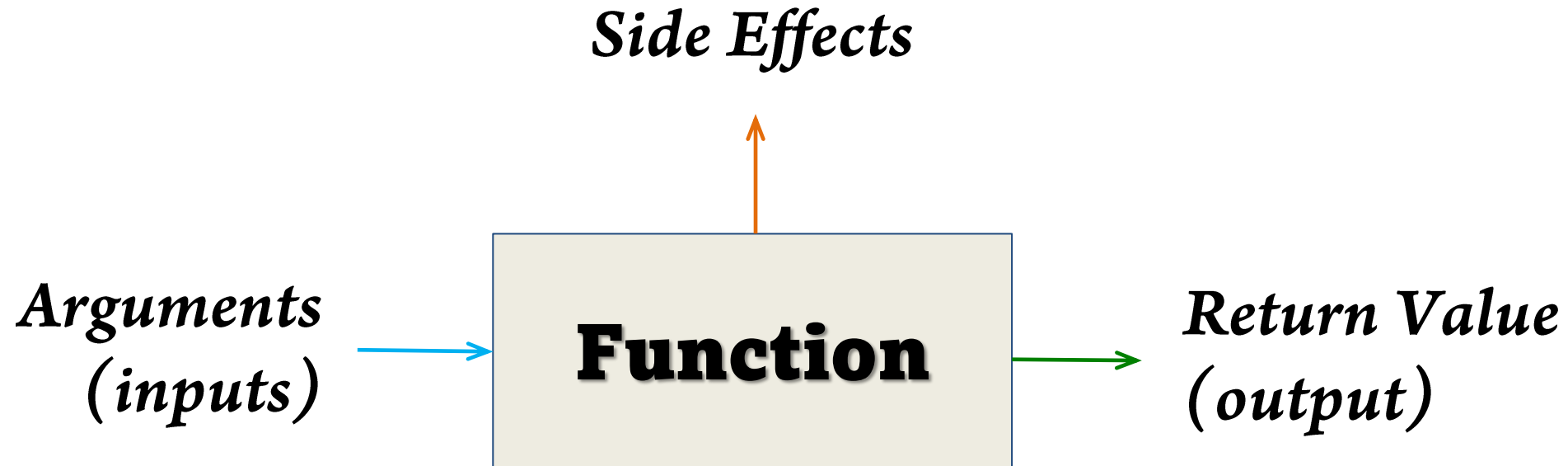
... almost as if this was happening

```
order = 'bagels'  
debit_card = '0000 1234 5678 1234'  
lunch = the_prepared_food
```

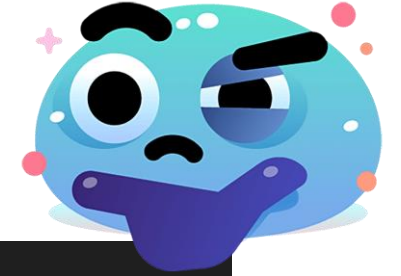
Functions: Vocabulary

- **Expression**: Code that **can become a value**
 - **Literal Expression**: an expression that cannot be simplified
- **Evaluate**: turn an expression into a **value**
- **Argument**: the value/expression we **send to a function**
- **Parameter**: name we use for the **variables in the function** (you see those in the **header**) that are **assigned to the arguments**
- **Call** (we also say '**Invoke**' or '**Execute**' the function): to use a function
- **Function Definition**: the code of the function (starts with “def”)
- **Return**: **exit the function with a value**
 - **Return Value**: **the value we exited with**
- **Side Effect**: actions that a function does (beyond returning something) **that affect the environment** outside the function

Function (inputs, outputs, side effects)



Q1



?

```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

Q1



Function
definition

```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

Q2



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

?

Q2



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

Function call

Q3



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

1.5 ?

Q3



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

argument

Q4

radius?



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

Q4

parameter



```
def cylinder_volume(radius, height):  
    area = 3 * radius ** 2  
    volume = area * height  
    return volume
```

```
v = cylinder_volume(1.5, 10)
```

Advice on Functions

- Use *informative* names
 - Typically, an action word/phrase
- Use a *comment* to describe the purpose of the function
- You usually *don't want* side effects
 - Don't use **print** unless you are required to, *return* a value instead
 - Don't use **input** unless you are required to, use a *parameter* instead
- *Save* useful functions in a file for *reuse* in the future (more on this later)

```

# goal of this function is to double a number
# input: the number that I want doubled
# output (return value): the doubled number
def my_function(x): # this is assigning the value of the
    # passed-in argument to the variable x
    x_doubled = x + x
    return x_doubled

# remember to typecast input if you want something other than a string
my_num = input("Enter a number: ")
doubled_my_num = my_function(float(my_num))
print("When we double", my_num, "we get", doubled_my_num)

#
#
# in the computer's memory, this is what's going on when we create
# variables....
#
# variable name | variable type | scope | variable address
# -----
# my_num | string | global | 0xFA724B
# doubled_my_num | float | global | 0xA73D91
# x | float | my_function | 0x453AB1

# and example of a function that doesn't return anything
def silly_function(): # no parameters
    print("this is just a print line in the silly function")
    return

```

Review this code on your own. It summarizes what we have covered. Don't hesitate to ask the TAs or the professor questions if you have any!

```

result = silly_function()
print("the silly function returned:", result)
# notice that the return is None

# since we ask the user to enter an integer often, we could create a
# function that would allow us to put the call to input, the type
# casting, and more (in the future, we'll think about input validation)
# all in one place
def get_int(the_prompt):
    # ... we could have more code here to make sure the number is valid
    in_num = int(input(the_prompt))
    return in_num

large = get_int("Enter a large number: ")
print(large * 2, "is larger than", large)

# you can always experiment with Python to learn the language better
# In this example, we are experimenting with types and operators
# try various operators on different pairings and orderings of
# a string and an int
#
str1 = "hello"
int1 = 9
result = str1 * int1 # swap the order, use different operators, etc.
print(type(result))
print(result)

# in the future, we'll discuss putting a function like this in a
# different # file than the one we are currently working on. Then we can
# call it
# anytime we want by using import

```



PYTHON DEMONSTRATION

Let's jump on PyCharm!

`function_basics.py`
`function_basics2.py`

Thinking about decisions and how to order conditions (generally: logic of code for a function)

- Let's consider the following question:
 - How can you tell if a given year is going to be a leap year?
- What is the formula? What is the algorithm to achieve this goal?

Thinking about decisions and how to order conditions (generally: logic of code for a function)

If a year is divisible by 4, it is a leap year, unless it is also divisible by 100, then it's not, unless it is also divisible by 400, then it is.

What does this look like in Python code?

Let's try to code this now

First, let's create some **test cases** so we will know if our function is working correctly

```
# These are some test cases that you can use for all of the Leap year  
# functions. Make sure to put them after the function definition.  
  
print(is_leap_year(2023))  # 2023 - no  
print(is_leap_year(2020))  # 2020 - yes  
print(is_leap_year(2100))  # 2100 - no  
print(is_leap_year(2000))  # 2000 - yes  
print(is_leap_year(0))     # 0 - yes
```

Remember that we can use the modulus operator, %, to determine divisibility

*# In order to make our if statements easy to read in the
following examples, we will use a **variable** to hold
True/False values for divisibility*

`div4 = ((x 4) = 0) # the year x is divisible by 4? - T/F`



What symbol should go here?

Remember that we can use the modulus operator, %, to determine divisibility

*# In order to make our if statements easy to read in the
following examples, we will use a **variable** to hold
True/False values for divisibility*

```
div4 = ((x % 4) = 0) # the year x is divisible by 4? - T/F
```



What symbol should go here?

Remember that we can use the modulus operator, %, to determine divisibility

```
# In order to make our if statements easy to read in the  
# following examples, we will use a variable to hold  
# True/False values for divisibility  
div4 = ((x % 4) = 0) # the year x is divisible by 4? - T/F
```

What's the bug here?

Remember that we can use the modulus operator, %, to determine divisibility

*# In order to make our if statements easy to read in the
following examples, we will use a **variable** to hold
True/False values for divisibility*

div4 = ((x % 4) = 0) *# the year x is divisible by 4? - T/F*


What's the bug here?



Remember that we can use the modulus operator, %, to determine divisibility

*# In order to make our if statements easy to read in the
following examples, we will use a **variable** to hold
True/False values for divisibility*

div4 = ((x % 4) == 0) *# the year x is divisible by 4? - T/F*



Here's the fix!

Remember that we can use the modulus operator, %, to determine divisibility

In order to make our if statements easy to read in the

*# following examples, we will use a **variable** to hold*

True/False values for divisibility

`div4 = ((x % 4) == 0)` *# the year x is divisible by 4? - T/F*

What is the type of **div4**?

Remember that we can use the modulus operator, %, to determine divisibility

In order to make our if statements easy to read in the

*# following examples, we will use a **variable** to hold*

True/False values for divisibility

`div4 = ((x % 4) == 0)` *# the year x is divisible by 4? - T/F*

`div100 = ((x % 100) == 0)` *# the year x is divisible by 100? - T/F*

`div400 = ((x % 400) == 0)` *# the year x is divisible by 400? - T/F*

Now let's start on the function itself

```
def is_leap_year(x):  
    # These 3 variables are used to hold True/False values for  
    # divisibility  
    div4 = ((x % 4) == 0) # the year is divisible by 4? - T/F  
    div100 = ((x % 100) == 0) # the year is divisible by 100? - T/F  
    div400 = ((x % 400) == 0) # the year is divisible by 400? - T/F  
  
    # "If a year is divisible by 4, it is a leap year, unless it is also  
    # divisible by 100, then it's not, unless it is also divisible by  
    # 400, then it is."  
  
    ...  
  
    return ...
```

Now let's start on the function itself

```
def is_leap_year(x):  
    # In this first version of the function, we have taken our text  
    # description -  
    #     "If a year is divisible by 4, it is a leap year, unless it is also  
    #     divisible by 100, then it's not, unless it is also divisible by  
    #     400, then it is."  
    # and converted it to a an equivalent <if> statement.  
  
    # These 3 variables are used to hold True/False values for  
    # divisibility  
    div4 = ((x % 4) == 0) # the year is divisible by 4? - T/F  
    div100 = ((x % 100) == 0) # the year is divisible by 100? - T/F  
    div400 = ((x % 400) == 0) # the year is divisible by 400? - T/F  
  
    if div4:  
        if div100:  
            if div400:  
                return True # if div4 and div100 and div400 → True  
            else:  
                return False # if div4 and div100 and not div400 → False  
        else:  
            return True # if div4 and not div100 → True  
    else:  
        return False # if not div4 → False
```

Strategy for writing **if** statements

- Start by considering a case where it is **easy to know the answer** (what case is it easiest to check for?)
- Then consider **the next case** where it will be easy to know the answer (*knowing you weren't in that first case*)
- If some cases have exceptions and others don't, **using the cases without exceptions first** may add *less complexity* to the program

These 4 functions are equivalent

1.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    if div4:  
        if div100:  
            if div400:  
                return True  
            else:  
                return False  
        else:  
            return True  
    else:  
        return False
```

3.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    if div400:  
        return True  
    elif div100:  
        return False  
    elif div4:  
        return True  
    else:  
        return False
```

2.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    return (div400 or ((not div100) and div4))
```

4.

```
def is_leap_year(x):  
    return ((x % 400) == 0) or ((not ((x % 100) == 0)) and ((x % 4) == 0))
```

These 4 functions are equivalent

1.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    if div4:  
        if div100:  
            if div400:  
                return True  
            else:  
                return False  
        else:  
            return True  
    else:  
        return False
```

3.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    if div400:  
        return True  
    elif div100:  
        return False  
    elif div4:  
        return True  
    else:  
        return False
```

Which one is easier to:
- read/understand?

2.

```
def is_leap_year(x):  
    div4 = ((x % 4) == 0)  
    div100 = ((x % 100) == 0)  
    div400 = ((x % 400) == 0)  
  
    return (div400 or ((not div100) and div4))
```

4.

```
def is_leap_year(x):  
    return ((x % 400) == 0) or ((not ((x % 100) == 0)) and ((x % 4) == 0))
```

These 4 functions are equivalent

1.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    if div4:
        if div100:
            if div400:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

3.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    if div400:
        return True
    elif div100:
        return False
    elif div4:
        return True
    else:
        return False
```

Which one is easier to:

- read/understand?
- troubleshoot/debug?

2.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    return (div400 or ((not div100) and div4))
```

4.

```
def is_leap_year(x):
    return ((x % 400) == 0) or ((not ((x % 100) == 0)) and ((x % 4) == 0))
```

These 4 functions are equivalent

1.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    if div4:
        if div100:
            if div400:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

3.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    if div400:
        return True
    elif div100:
        return False
    elif div4:
        return True
    else:
        return False
```

Which one is easier to:

- read/understand?
- troubleshoot/debug?
- write?

2.

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)

    return (div400 or ((not div100) and div4))
```

4.

```
def is_leap_year(x):
    return ((x % 400) == 0) or ((not ((x % 100) == 0)) and ((x % 4) == 0))
```

Activity on Conditionals

- In **pairs** or groups **up to three** work on the following activity.
- **function_basics_ica.py**
- *Practice writing code.*
Practice commenting your code, too.

Remember to **check-in** with a TA before leaving class today!

In-Class “lab” Activity!

Given below are several versions of a function that calculates if a
given year is a leap year. Although each function produces the correct
answer, they are written very differently.

These are some test cases that you can use for all of the leap year
functions. Make sure to put them after the function definition.

```
print(is_leap_year(2023)) # 2023 - no
print(is_leap_year(2020)) # 2020 - yes
print(is_leap_year(2100)) # 2100 - no
print(is_leap_year(2000)) # 2000 - yes
print(is_leap_year(0))    # 0 - yes
```

In order to make our if statements easy to read in the following
examples, we will use these 3 variables to hold True/False values for
divisibility

```
div4 = ((x % 4) == 0) # the year is divisible by 4? - T/F
div100 = ((x % 100) == 0) # the year is divisible by 100? - T/F
div400 = ((x % 400) == 0) # the year is divisible by 400? - T/F
```

In this first version of the function, we have taken our text
description -
"If a year is divisible by 4, it is a leap year, unless it is also
divisible by 100, then it's not, unless it is also divisible by
400, then it is."
and converted it to an equivalent <if> statement.

```
def is_leap_year(x):
    # These 3 variables are used to hold True/False values for
    # divisibility
    div4 = ((x % 4) == 0) # the year is divisible by 4? - T/F
    div100 = ((x % 100) == 0) # the year is divisible by 100? - T/F
    div400 = ((x % 400) == 0) # the year is divisible by 400? - T/F

    if div4:
        if div100:
            if div400:
                return True
            else:
                return False
        else:
            return True
    else:
        return False
```

Review this code on your own. It
summarizes what we have covered.
Don't hesitate to ask the TAs or the
professor questions if you have any!

```
# In this next version of the function, we started with the most
# constrained case:
# Any year that is evenly divisible by 400 - we know that is a leap
# year and the other calculations won't matter for that year.
# Then we take the next most constrained case and continue on until all
# cases have been considered.
```

```
#
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)
```

```
    if div400:
        return True
    elif div100:
        return False
    elif div4:
        return True
    else:
        return False
```

```
# In this next version of the function, we show how we could reduce the
# if statement in to one boolean expression that uses boolean operators.
# While this function is shorter, it is likely more difficult to read.
```

```
def is_leap_year(x):
    div4 = ((x % 4) == 0)
    div100 = ((x % 100) == 0)
    div400 = ((x % 400) == 0)
```

```
    # In the following line, if div400 is True, then it doesn't matter
    # what is on the other side of the <or>, the whole expression will
    # be True. But if div400 is not True, both div4 and (not div100)
    # must be True to make the expression True.
    return (div400 or ((not div100) and div4))
```

```
# This last version of the function takes the previous version and
# replaces the divisibility variables with the expressions that they
# were assigned to. This version works correctly and is extremely short,
# but very difficult to read. I would not use it because it would be
# difficult to debug if it contained an error. And it would be difficult
# for someone else to understand.
```

```
def is_leap_year(x):
    return ((x % 400) == 0) or ((not ((x % 100) == 0)) and ((x % 4) == 0))
```

Review this code on your own. It summarizes what we have covered. Don't hesitate to ask the TAs or the professor questions if you have any!

Quick Review

= vs == (1 equal sign vs 2 equal signs)

- = **Assignment**
 - `x = 5`
 - The value 5 is assigned to the variable x
- == **Checking equality**
 - `x == 5`
 - Is the value of variable x equal to 5?
 - Evaluates to True or False

Reminder: CS Laptop Loaner Program

- This course requires students to have a **laptop**
- I realize that not everybody might have one (nor necessarily need one for their desired major / path...)
- If you do not have a laptop for any reason... *not to worry!*
- The CS department's Systems staff has a notebook / laptop loaner program and will be able to loan you a notebook / laptop computer for the duration of the semester if you don't have one or if you cannot afford one.
 - Also available if your laptop is broken and under repair, we can arrange for you to receive a loaner laptop for a week or two until your own laptop is fixed

Interested? Link: https://www.cs.virginia.edu/wiki/doku.php?id=cs_laptop_loaner

I am happy to be your sponsor. Please let me know.