# Announcements

- Homework 1 due Friday night
  - Problems given in pdf
  - Link to add template repository is also in pdf
    - Download the repository to your own system to work on locally!
  - Some extra guidelines in the repository README
  - Demo of automatic testing here in a moment
- Polling: `rembold-class.ddns.net`

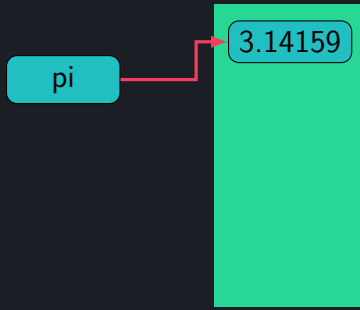
# Testing Demo

- There is some automatic testing built in to the homework repositories
- Requires no extra effort on your part aside from not messing with special code I've written
- Gives you some idea if your code is generally behaving as intended
- Works on the *entire* repository, so everything needs to be good to get a green passing badge on the README
- Can see a breakdown of which tests failed by following details (see demo)
- Can also view tests in VSCode
  - Select PYTEST the first time when prompted, then menu on left

# All things must change

- You can rebind variables using a new assignment statement
- Old value gets lost
- Variables only change upon assignment. Changing something that a variable depends on does *not* update that variable.

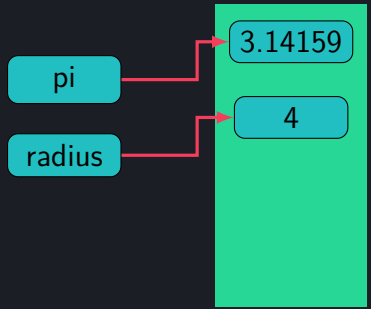
```
pi = 3.14159
radius = 4
circ = 2 * pi * radius
radius = radius + 2
```



# All things must change

- You can rebind variables using a new assignment statement
- Old value gets lost
- Variables only change upon assignment. Changing something that a variable depends on does *not* update that variable.

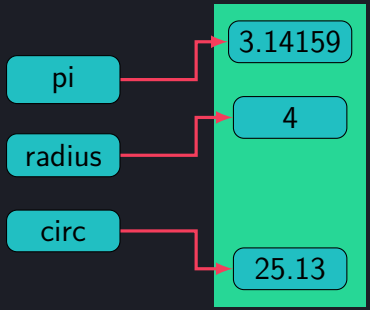
```
pi = 3.14159
radius = 4
circ = 2 * pi * radius
radius = radius + 2
```



# All things must change

- You can rebind variables using a new assignment statement
- Old value gets lost
- Variables only change upon assignment. Changing something that a variable depends on does *not* update that variable.

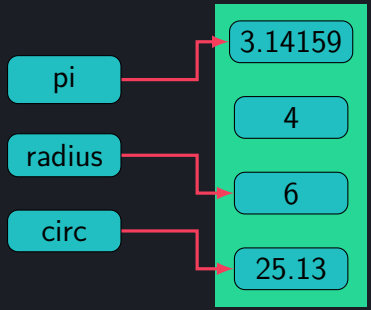
```
pi = 3.14159
radius = 4
circ = 2 * pi * radius
radius = radius + 2
```



# All things must change

- You can rebind variables using a new assignment statement
- Old value gets lost
- Variables only change upon assignment. Changing something that a variable depends on does *not* update that variable.

```
pi = 3.14159
radius = 4
circ = 2 * pi * radius
radius = radius + 2
```



# Two other #Comments

- You can assign multiple variables at the same time

```
x, y = 5, 10
```

  - This can be useful for swapping variables, since everything gets evaluated before the bindings are assigned

```
x, y = y, x
```

- Use comments in your code to explain otherwise confusing or unclear sections

```
#This is a comment
#The interpreter will completely ignore these
#lines while the code is being executed
4 + 5
```

# Understanding Check

What is the final printed value of `A` in the code below?

```
1 A = 10
2 B = 4
3 C = A * B
4 A = C - A
5 # B = A
6 A, B, C = C, A, B
```

A) 4

B) 10

C) 30

D) 40

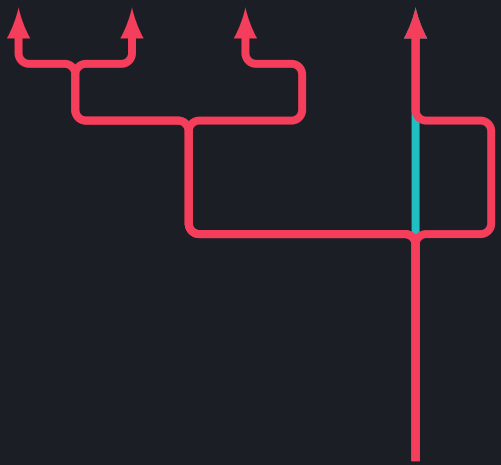
**Solution:** 40

# Moving beyond making calculators

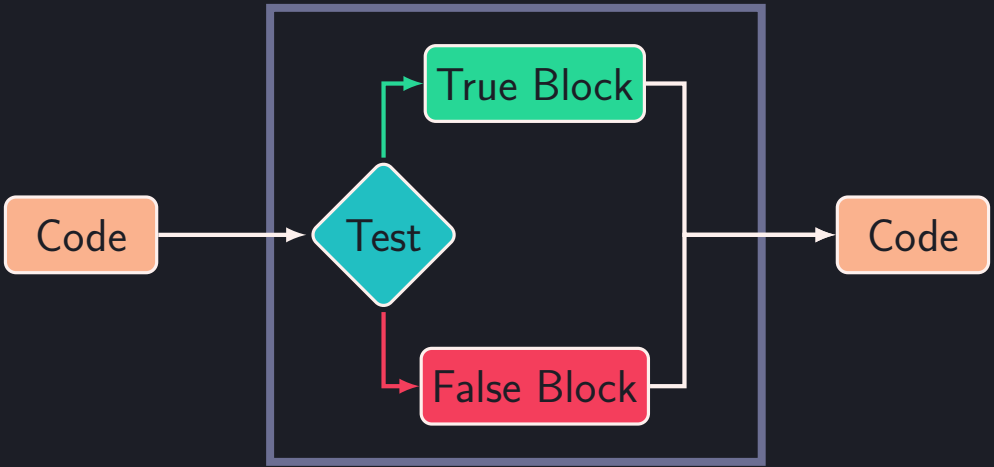
- Recall that a requirement of an algorithm is that it has some sort of flow control
- Thus far, everything we've done or looked at proceeds in a straight line
  - Simple, but inherently boring
  - Not much you can do with it beyond basic calculations
- Want to talk now about how to write branching programs

# Moving beyond making calculators

- Recall that a requirement of an algorithm is that it has some sort of flow control
- Thus far, everything we've done or looked at proceeds in a straight line
  - Simple, but inherently boring
  - Not much you can do with it beyond basic calculations
- Want to talk now about how to write branching programs



# The Conditional



# Python Conditional Syntax

In Python, the basic conditional looks like:

```
if boolean_expression:
    # True block of code
    # Code to run if the test is true
else:
    # False block of code
    # Code to run if the test is false
```

- Things to note:
  - A colon : ends each of the `if` and `else` lines
  - True and False blocks are indented
  - You do not need an `else` portion if your desired false block is empty
  - Boolean expression can be anything that evaluates to True/False

# Tests

- Tests can be any expression that returns a Boolean, but common ones are:
  - `==` check if two things are equal
  - `>` check if something is greater than the other
  - `<` check if something is less than the other
  - `>=` check if something is greater than or equal to the other
  - `<=` check if something is less than or equal to the other
  - `!=` check if two things are not equal

# Compounds and Nests

You can check multiple conditions in a variety of ways

- Nested Conditionals

```
if num%3 == 0:
  if num%4 == 0:
    print("Divisible by 3
and 4")
  else:
    print("Divisible by 3")
```

- Compound Conditionals

```
if num%3==0 and num%4==0:
   print("Divisible by 3
and 4")
```

# Mr Elif

- A common issue that arises is when one condition is false (so you'd move to the false block) but then you immediately want to check another condition
- Python has a shorthand for this: the `elif`

```
if animal == "Dog":
    print("Woof!")
elif animal == "Cat":
    print("Meow!")
elif animal == "Bird":
    print("Tweet!")
elif animal == "Fox":
    print("Ring-ding-ding-ding-dingeringeding!")
```

- Usually want mutually exclusive things: something can't be both

# Understanding Check

```
animal = "shark"
number = 5

if animal == "kitten":
    if number > 10:
        print("I am in heaven!")
    else:
        print("I am still happy!")
elif animal == "goat" and number > 5:
    print("This is getting excessive.")
if animal == "shark":
    if number > 0:
        print("This is the worst day.")
else:
    print("Today is an ok day.")
```

What will be printed to the screen if the code to the left is run?

A) This is getting excessive.
B) Today is an ok day.
C) I am still happy!
D) Nothing printed to screen

**Solution:** Nothing printed to screen

# String Theory

- Our first non-scalar object type
- Comprised of multiple characters
- Enclosed in either single or double quotes

```
"This is a string!"
'This is also a string.'
```

# Stringy Operations

- `'a' + 'a'` concatenates strings to give `'aa'`
- `3 * 'ha'` repeats strings to give `'hahaha'`
- `len('fish')` will give the length of the string (4 here)
- Comparisons check that all parts of the string match
  - `'fish' == 'Fish'` will return `False`
- Be careful of < or > comparisons of strings
  - Generally work alphabetically
  - But any capital letter at the *start* is "less" than a lowercase letter

# Getting Your Input

- We already have `print()` for writing things to the terminal
- How can we give the program some sort of input short of editing the code?
  - One method is the aptly named `input()` function
  - Gives you a prompt at the terminal to enter something

```
guess = input('Guess any number between 1 and 10: ')
```

  - Always returns a string, so further conversion may be necessary