Python for Absolute Beginners

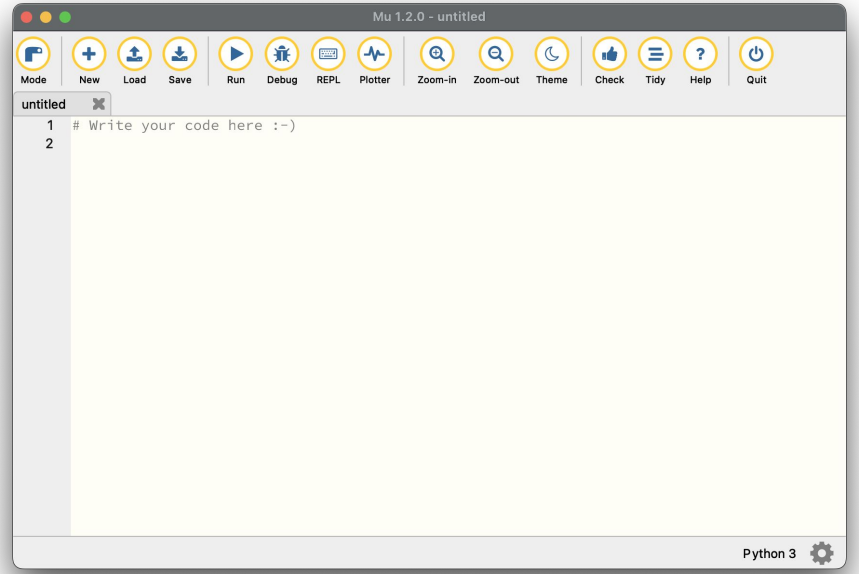
Al Sweigart

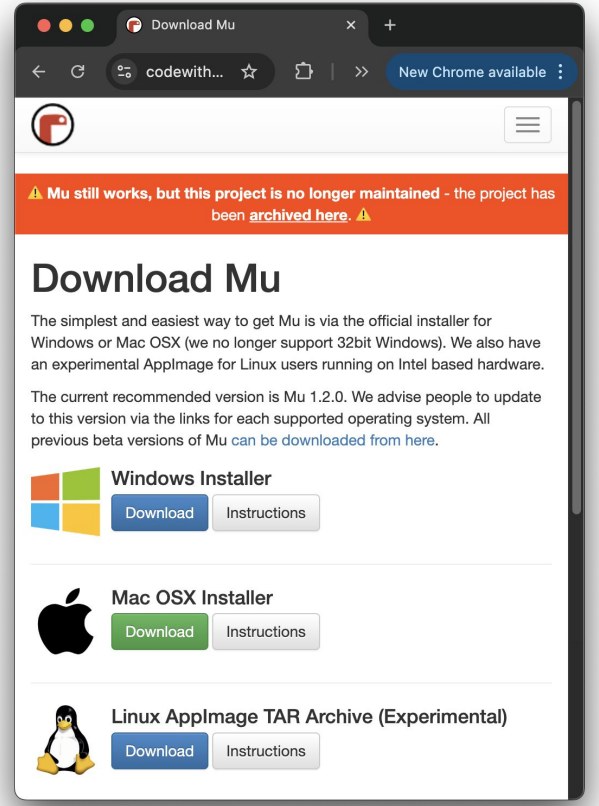
PyCon US 2026 Tutorial

https://inventwithpython.com/pyconus2026

Last Second Installing of Mu If You Haven't Done That Yet

https://codewith.mu/





Who Am I?

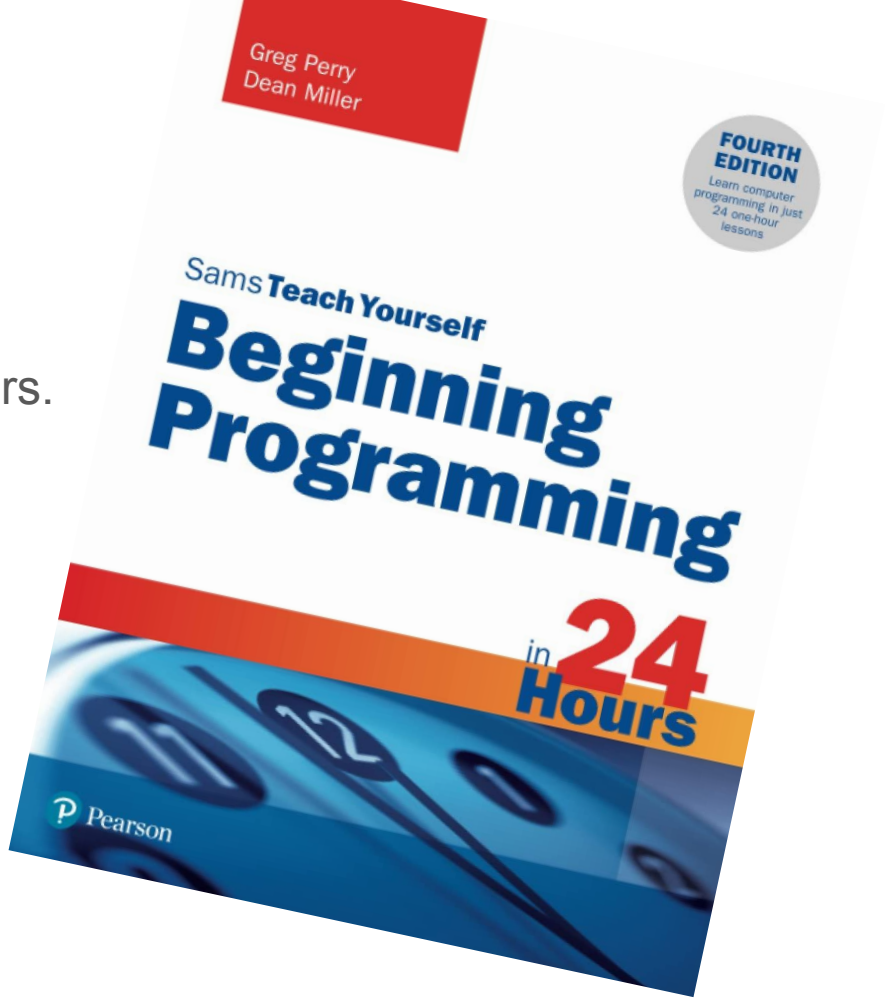
- Al Sweigart (last name rhymes with “why dirt”)
- I wrote *Automate the Boring Stuff with Python*
- I wrote a bunch of other programming books
- All the books are free at https://inventwithpython.com

BAD NEWS:

You cannot learn to program in 3 ½ hours.

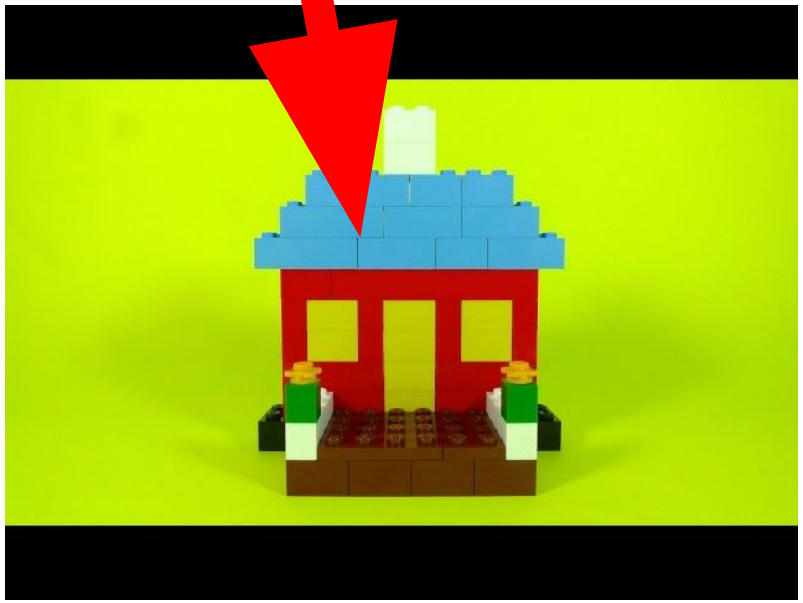
GOOD NEWS:

You can get a good start in 3 ½ hours.



https://scrollart.org

We are doing this.



Not this.

My Pet Peeve

“What you should really teach is critical thinking.”

What Is Python? (language)

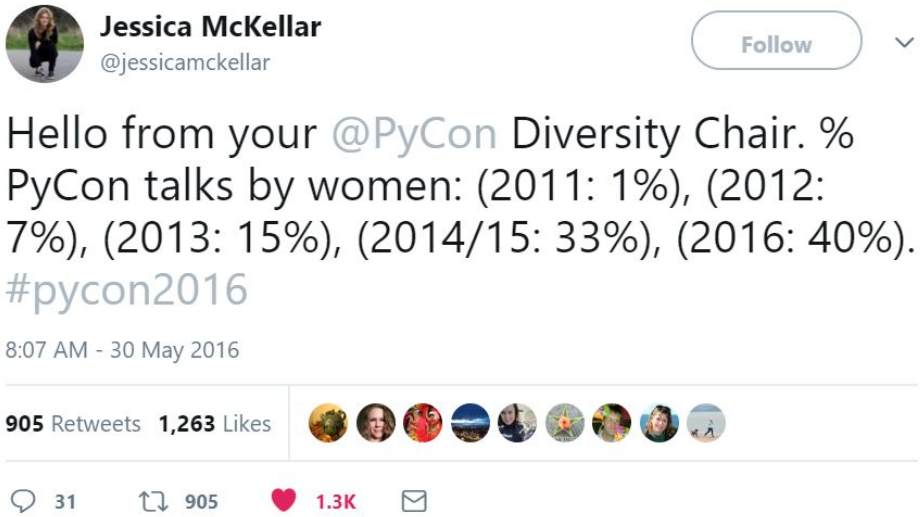
- Python is a programming language - “I wrote a program in Python.”
- Python is free, open source, and community-maintained
- Python became really popular in the 2010s
- Python is surprisingly old: the first release was made in 1991
- Python is the best first programming language to learn

What Is Python? (software)

- Python is the free interpreter software that runs Python code - “Python ran this program” “Python gave me an error message”
- You download the Python interpreter from https://python.org
- (The Mu code editor bundles Python with it.)

What is Python? (community and ecosystem)

- “The hardest part of tech is never the tech.”
- “Come for the language, stay for the community”



You Will Not Break or Damage The Computer

- Nothing in this tutorial will damage your computer
- Error messages just mean the Python interpreter didn't understand your code, so it stopped
- "Crashing" just means the Python interpreter didn't understand your code, so it stopped running the program
- **Error message worksheet: Gotta Catch 'Em All!**
- DON'T PANIC!

Programming is Something You Do

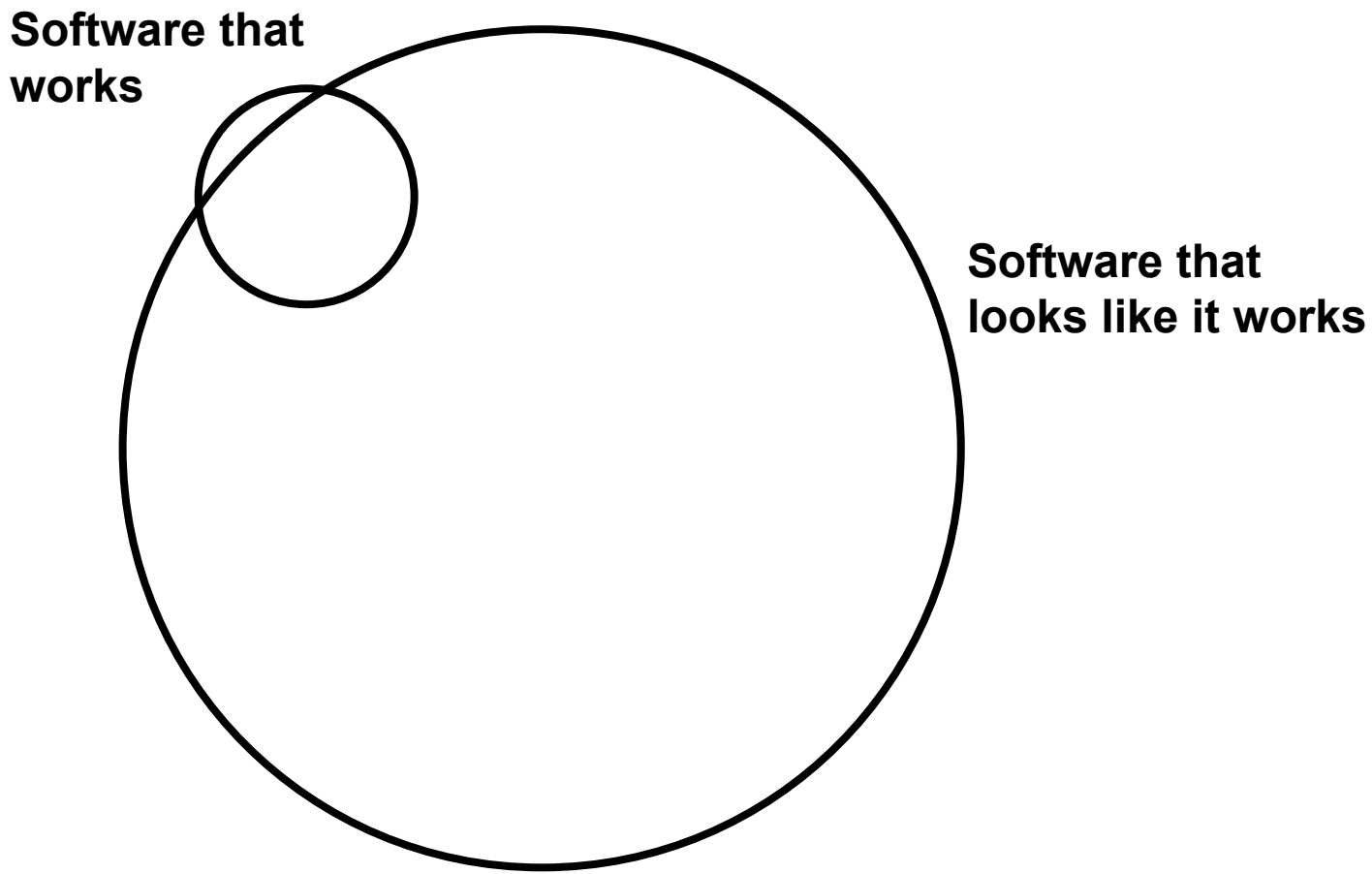
- Re-typing the source code forces you to slow down and get “muscle memory”
- **Slow is smooth and smooth is fast**

- Watching videos or reading blog posts is forgettable
- Copy/pasting code keeps you from paying attention to it
- Having AI generate code keeps you from paying attention to it

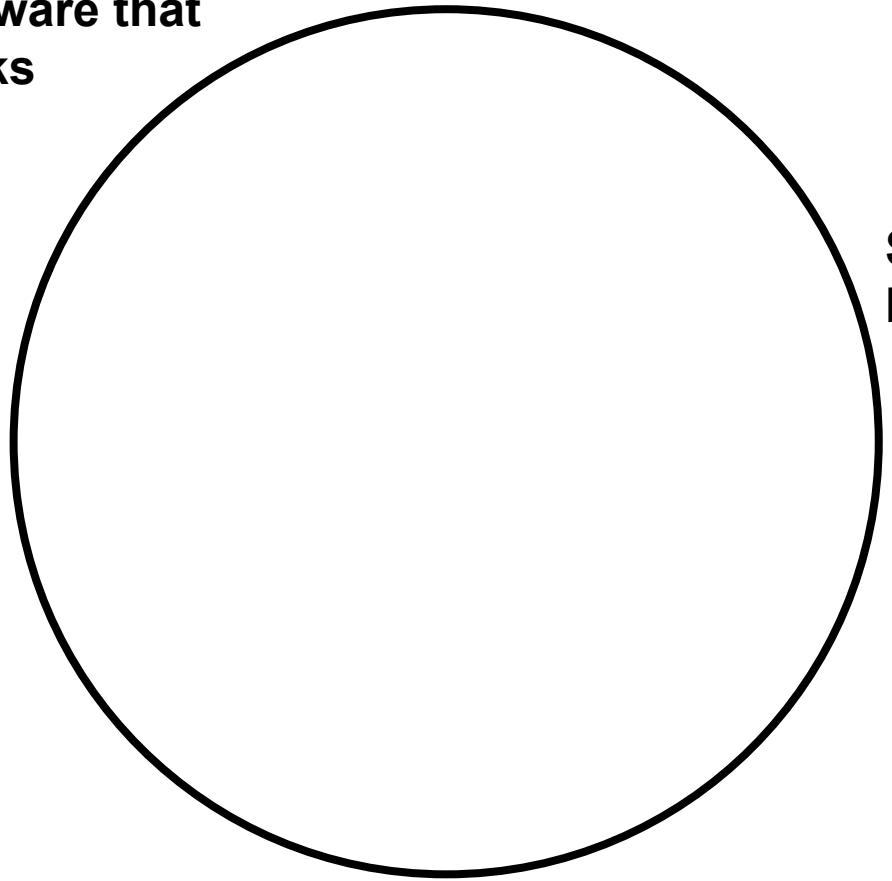
- Googling and looking things up is not “cheating”

AI’s Thoughts On Using AI To Learn To Code

- Don’t.


Software that
works
Software that
looks like it works

Software that works

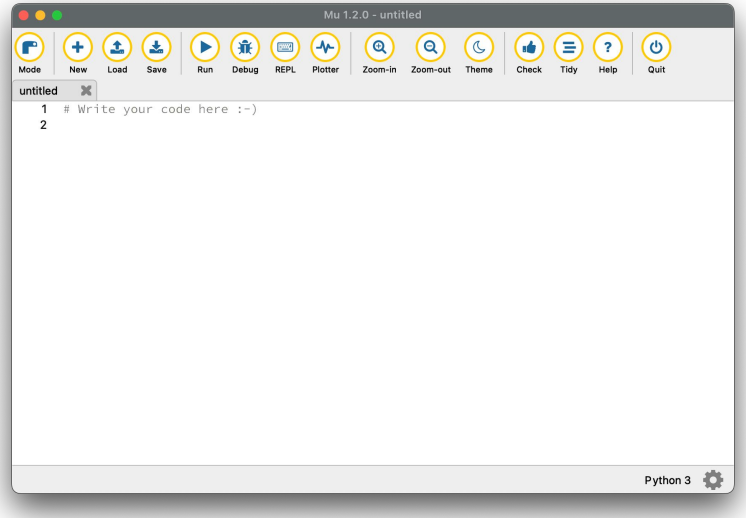


Software that looks like it works

LET’S BEGIN

Mu, the Code Editor App

- WORKBOOK p2: “Mu, the Code Editor App”
- GREEN FLAG: When you’ve read page 2 & 3 and have Mu opened
- If you can’t open it, DON’T PANIC. Open https://pythontutor.com



Hello, world!

- WORKBOOK p4: “Example Program: Hello, World!”
- GREEN FLAG: When you’ve run hello.py by clicking the Run button.

- If you get an error message, double check for typos.
- Get our attention or put up the RED FLAG if you have a question.
- If Mu isn’t working, DON’T PANIC. Enter the program into https://pythontutor.com

Hello, world!

- WORKBOOK p5: Run the “Hello, World!” Program Again With Silly Input
- GREEN FLAG: When you’ve run the program with silly input

Comments

- Comments are text in the source code that the Python interpreter ignores
- Programmers use comments to make notes or explain the code
- Comments begin with # and go to the end of the line
- It’s good style to put one space between the # and the comment text.
- `#Don’t do this.`

```
>>> # This si a comment.
```

print() TODO

Function Calls and Arguments

- Functions are like mini-programs in your program
- We type their names with parentheses to mark them as functions: print()
- You can run the code in them by *calling* them
- Function calls can accept zero or more input data called *arguments*
- TODO two args, user_name doesn’t have quotes

Rules for Variable Names TODO cut

- Case-sensitive: `SPAM`, `spam`, `Spam` are separate variables
- Can only have letters, numbers, and _ underscores
- Cannot begin with a number
- Must be one word; cannot have spaces (use an underscore instead)
- Cannot be a Python keyword (`if`, `else`, `while`, `for`, `return`, `True`, `False`; we'll see these later)

Good Advice: Make the variable name describe the data the variable contains. Labelling your moving boxes as "stuff" isn't helpful.

Hello, world!

- WORKBOOK p6: Flow Chart for the “Hello, World!” Example Program
- WORKBOOK p7: Run the “Hello, World!” Program Under the Debugger
- GREEN FLAG: When you’re done running the program under the debugger

REPL - Calculator Stuff

- WORKBOOK p9: “Calculator Stuff” in the REPL / Interactive Shell
- GREENFLAG: When you’ve filled out the page

Expressions, Values, and Operators

- *Values* are pieces of data, such as numbers or text
- *Operators* like + - * / act on data
- *Expressions* are instructions made up of values and operators that *evaluate* to a single value
- Expressions can look like math problems like 2 + 2, but there are non-math expressions too
- Evaluation animation: https://inventwithpython.com/pyconus2026/evalexpr.html

Strings and String Operations

- Strings are values that represent text
- String values begin and end with a single quote (you can also use double)

- The + operator can *concatenate* two string values together
- The * operator can *replicate* a string and an int value

REPL - Strings

- WORKBOOK p10: String Concatenation and String Replication in the REPL / Interactive Shell
- GREEN FLAG: When you’ve filled out the page

Boolean Values and Comparison Operators

- The Boolean data type has exactly two values: True and False
- There are six comparison operators: `==` `!=` `<` `<=` `>` `>=`
- Expressions with comparison operators evaluate to Boolean values:
- 42 < 10 evaluates to False

REPL - Boolean Values

- WORKBOOK p11: Boolean Values and the Comparison Operators in the REPL / Interactive Shell
- GREEN FLAG: When you've filled out the page

Temperature Conversion

- WORKBOOK p12: Writing a Math Equation as Python Code
- GREEN FLAG: When you've filled in the blanks and entered the code into the REPL.

Temperature Conversion

- WORKBOOK p13: Example Program: Fahrenheit / Celsius Temperature Conversion
- GREEN FLAG: When you've run the program a couple times with different temperatures to convert

`if` Statements

Sometimes you want code to run only if a certain condition is true, that is, if an
expression evaluates to `True`. For these cases, you want an `if` statement, which

- The `if` keyword (must be lowercase)
- A condition (an expression that evaluates to `True` or `False`)
- A colon
- An indented block of code starting on the next line. The “block” is made of the lines of code with the same level of indentation. The block ends when the indentation ends.

```
if name == 'Albert':
    print('Hello')
```

Temperature Conversion

- WORKBOOK p14: Flow Chart for the Temperature Conversion Program
- WORKBOOK p15: Run the “Temperature Conversion” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger a couple times.

The round() Function

- WORKBOOK p16: The round() Function
- WORKBOOK p17: Add the round() Function to the Temperature Conversion Program
- GREEN FLAG: When you've added the round() function to the program and re-run it.

Importing Modules

- Python comes with several functions to do useful and common actions. It would slow your program down if Python loaded ALL of them for EVERY program, so some functions exist inside of modules that you can import into your program with import statements.
- `import random`

Importing Modules

- WORKBOOK p18: Importing Modules and the random.randint() and sys.exit() Functions
- GREEN FLAG: When you've filled in the blanks on the page.

The % Modulo Operator

- WORKBOOK p19: Calculating If a Number is Even or Odd
- GREEN FLAG: When you have filled in the blanks on the page.

Boolean Operators and Truth Tables

- WORKBOOK p20: Boolean Operators and Truth Tables
- GREEN FLAG: When you've filled in the blanks on the page

else Statements

An else statement can optionally follow an if statement. Think of it as “If this condition is true, run this block, or else if the condition was false, run this other block.

```
if some_variable == 5:
    # block of code
else:
    # other block of code
```

One and only one of the code blocks is run.

Cho Han Dice Game

- WORKBOOK p21: Example Program: Cho Han Dice Game
- GREEN FLAG: When you've run the program a few times

Cho Han Dice Game

- WORKBOOK p22: Flow Chart for the Cho Han Dice Game Program
- WORKBOOK p23: Run the “Cho Han Dice” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

while Loops

- while Loops look very similar to if statements, except they have the `while` keyword instead of the `if` keyword, and the execution loops back to the start of the `while` statement when it reaches the end.
- A while statement says, “While this condition is True, run the code in this block”

“Your Name” Program

- WORKBOOK p24: while Loops and the Example Program: “Your Name” Joke
- GREEN FLAG: When you’ve run the program

“Your Name” Program

- WORKBOOK p25: Flow Chart for the “Your Name” Program
- WORKBOOK p26: Run the “Your Name” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

Infinite Loops and break statements

- If you use `while True:` then the condition of the `while` statement is ALWAYS true and you have an infinite loop.
- A `break` statement causes the execution to leave (“break out of”) the `while` loop.

“Your Name” 2 Program

- WORKBOOK p27: Infinite Loops, break Statements, and the “Your Name 2” Program
- GREEN FLAG: When you’ve run the program

“Your Name” 2 Program

- WORKBOOK p28: Flow Chart for the “Your Name 2” Program (with a break Statement)
- WORKBOOK p29: Run the “Your Name 2” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

for Loops

- A `while` loop says “run this block of code while this condition is true”
- A `for` loop says “run this block of code a specific number of times”
- `for i in range(10):` runs the code block ten times
- The i variable starts with value 0, then 1, then 2, up to (but not including) 10

Clever Carl

- WORKBOOK p30: Example Program: Clever Carl
- WORKBOOK p31: An Equivalent Clever Carl Program Without the for Loop
- GREEN FLAG: When you've run the program and read through page 31

Clever Carl

- WORKBOOK p32: Flow Chart for the “Clever Carl” Program
- WORKBOOK p33: Break Points and Running the “Clever Carl” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger with break points

end= in print()

- By default, a newline character (typed in code as `'\n'`) is printed at the end of the string you pass to `print()`
- We can change this to a space so that the text fills up horizontally instead of vertically: `print('Hello', end=' ')` and `print('world!')` prints `Hello world!` on the same line.
- `end=' '` is called a keyword argument

The range() Function

- The `range()` function can be called with one, two, or three arguments.
- `range(`*`stop`*`)` starts `i` at `0` and goes up to but not including *stop*
- `range(`*`start`*`, `*`stop`*`)` starts `i` at *start* and goes up to but not including *stop*
- `range(`*`start`*`, `*`stop`*`, `*`step`*`)` starts `i` at *start* and goes up to but not including *stop*, increasing `i` by *step* instead of by 1
- `range(10)` is the same as `range(0, 10)` or `range(0, 10, 1)`

The range() Function

- WORKBOOK p34: The range() Function
- GREEN FLAG: When you've filled in the blanks

Guess the Number

- WORKBOOK p35: Example Program: Guess the Number
- GREEN FLAG: When you've run the program

Guess the Number

- WORKBOOK p36: Flow Chart for the “Guess the Number” Program
- WORKBOOK p37: Run the “Guess the Number” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

Nested Loops

- You can put loops inside of other loops!
- Think of it as the seconds hand of a clock must go in a full circle to move the minutes hand over by 1, and the minutes hand must go in a full circle to move the hour hand by 1.
- https://inventwithpython.com/nested-loops.html

Clock Hands 1

- WORKBOOK p38: Example Program: Clock Hands 1
- GREEN FLAG: When you've run the program

Clock Hands 1

- WORKBOOK p39: Flow Chart for the “Clock Hands 1” Program
- WORKBOOK p40: Break Points and Running the “Clock Hands 1” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

Clock Hands 2

- WORKBOOK p41: Example Program: Clock Hands 2
- GREEN FLAG: When you've run the program

Clock Hands 2

- WORKBOOK p42: Flow Chart for the “Clock Hands 2” Program
- WORKBOOK p43: Break Points and Running the “Clock Hands 2” Program Under the Debugger
- GREEN FLAG: When you’ve run the program under the debugger

Defining Your Own Functions

- You can define your own functions with a def statement:

```
def function_name():
    # some code goes here

function_name()  # calls the function
```

- This prevents you from copy/pasting the same code. Define the function once and call it multiple times.
- If you need to change the code, there’s only one place that needs to be changed.

Starfield Scroll Art

Scroll art programs produce interesting animations using print() inside of infinite loops. Other scroll art examples: https://scrollart.org

- WORKBOOK p44: Example Program: Starfield Scroll Art
- GREEN FLAG: When you've run the program