Python Camp — Day 1
Python Basics

David Pouliot
Southern Oregon University

Today

- Variables and value types (string, int, float, bool)
- Talking to the user: `input()` and `print()`
- f-strings — putting values inside text
- Making decisions: `if / elif / else`
- Repeating: `while` and `for` loops
- Lists — holding many values at once

Two projects today: **Mad Libs** and a **Number Guessing Game**.

About Me

- 6 years teaching CS at SOU
- Research Interests:
  - Applied Cryptography, Searchable Encryption
  - Symbolic Execution
  - Practical Security Solutions
- Hobbies are furniture building and whitewater kayaking

About You

- Your Name
- Something about you (Optional, choose zero or more)
  - Dream job
  - Interests
  - Favorite food
  - Hobbies
  - ?

Variables & Strings

A variable is a labeled box

You put a value in; the name lets you get it back later.

```
name = "Ada"
age = 16
print(name)
print(age)
```

- Text goes in quotes: `"Ada"` — that's a string.
- Numbers don't: `16`.

Every value has a type

The type decides what you're allowed to do with a value.

Type	What it is	Example
`str`	text (in quotes)	`"Ada"`
`int`	whole number	`16`
`float`	number with decimals	`3.14`
`bool`	true or false	`True`

Not sure? Ask Python with `type()`:

```
1 print(type("Ada"))    # <class 'str'>
2 print(type(16))       # <class 'int'>
```

Types matter: + does two different things

```
print(3 + 4)          # 7      -> adds the numbers
print("3" + "4")      # 34     -> glues the strings
print("3" + 4)        # ERROR! -> can't mix the two
```

- `input()` always gives a string — even `"16"`.
- Convert when you need a number: `int("16")` or `float("3.14")`.
- Go the other way with `str(16)`.

Most Day 1 bugs are a number that's secretly a string. Check the type!

Input and print

`input()` always gives you back a string.

```
name = input("What is your name? ")
print("Hello, " + name)
```

Need a number? Convert it with `int()`:

```
age = int(input("How old are you? "))
print(age + 1)
```

f-strings: the easy way to build text

Put an `f` before the quote, then drop variables in {`curly braces`}.

```
name = "Ada"
age = 16
print(f"{name} is {age} years old.")
```

f-strings are your best friend all week. Use them everywhere.

Project: Mad Libs

Mad Libs

Ask for a few words, then drop them into a silly story.

```
noun = input("Give me a noun: ")
verb = input("Give me a verb: ")
adj  = input("Give me an adjective: ")

print(f"The {adj} {noun} loves to {verb} all day.")
```

Your turn: make the story longer and funnier.

Decisions & Loops

if / elif / else

Python runs the first branch whose test is true.

```
score = int(input("Your score: "))
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
else:
    print("Keep practicing!")
```

Indentation decides what's inside the `if`

The indented line belongs to the `if`. The line lined up with `if` is outside it.

```
1 score = int(input("Your score: "))
2
3 if score >= 90:
4     print("You aced it!")    # INDENTED  -> runs only if score >= 90
5 print("Thanks for playing") # NOT indented -> runs no matter what
```

- Type 95: you see **both** lines.
- Type 70: you see **only** "Thanks for playing".

while loops

Keep going as long as the test stays true.

```
count = 1
while count <= 5:
    print(count)
    count = count + 1
```

Change something inside the loop, or it runs forever!

Lists

A list holds many things

One name, many values — written in `[square brackets]`.

```
foods = ["pizza", "tacos", "sushi"]

print(foods[0])        # pizza  (counting starts at 0!)
print(len(foods))      # 3      (how many items)

foods.append("ramen") # add one to the end
```

- The first item is number 0, not 1.
- `len()` tells you how many are in the list.

Looping over a list with `for`

A `for` loop hands you each item, one at a time.

```
foods = ["pizza", "tacos", "sushi"]

# each time through, 'food' becomes the NEXT item in the list
for food in foods:       # pass 1: food = "pizza"
    print(f"I love {food}!")   # pass 2: food = "tacos"
                               # pass 3: food = "sushi", then stop
# prints: I love pizza!  /  I love tacos!  /  I love sushi!
```

- Use `for` when you know the list to walk through.
- Use `while` when you're waiting for something to happen.

Lists + random = Magic 8-Ball

`random.choice()` picks one item from a list for you.

```
import random

answers = ["Yes", "No", "Ask again later", "Definitely!"]

question = input("Ask the Magic 8-Ball: ")
print(random.choice(answers))
```

Your turn: add more answers. Make some silly.

Project: Number Guessing Game

Number Guessing Game — your turn to build it

Goal: the computer picks a secret number 1–100. The player keeps guessing until they get it, and each wrong guess gets a hint.

```
import random
secret = random.randint(1, 100)   # the number to guess

# TODO -- you write this part:
#   keep asking until the guess is right
#     - read a guess with  int(input("Guess: "))
#     - if the guess is too low,  print "Too low!"
#     - if the guess is too high, print "Too high!"
#   when it finally matches, print "You got it!"
```

Stretch the guessing game

- Count the guesses and print a score at the end
- Only allow 7 guesses — then "you lose"
- Keep a list of every guess, print it at the end
- Let the computer guess *your* number instead

Finished early? Grab a stretch goal. Stuck? Raise your hand.

Pick-Your-Own Projects

More project options — pick any

Starter — everyone finishes

- Fortune Teller / Magic 8-Ball
- Tip / pizza-split calculator
- Times-table printer

Standard — pick one

- Choose-your-own-adventure
- Password strength checker
- Rock–Paper–Scissors

Boss challenge — if you're flying: Caesar cipher (shift each letter by N) · Hangman

No wrong choice. Full starter code for these is on the camp website.

Idea starters

```python
# Rock-Paper-Scissors: let the computer pick
import random
options = ["rock", "paper", "scissors"]
computer = random.choice(options)

# Password checker: string tests return True / False
pw = input("Pick a password: ")
print(len(pw) >= 8)              # long enough?
print(any(c.isdigit() for c in pw))   # has a number?
```

Save Your Work: Git & GitHub

Why version control?

A **version control system** saves snapshots of your code as you work — so a bad edit can never wipe out your progress.

- **Undo** — jump back to any earlier snapshot that worked
- **Backup** — your code lives online, not just on one computer
- **History** — see what changed, and when

Git is the tool that takes the snapshots. **GitHub** is the website that stores them online. Almost every programmer uses both.

Start on GitHub: create, then clone

Step 1 — create the repo. Make a free account at `github.com`, click **New repository**, and name it (e.g. `my-code`).

Step 2 — clone it. Copy the repo's URL, then clone to pull a copy down onto your computer:

```
git clone https://github.com/you/my-code.git
```

GitHub **requires two-factor authentication (2FA)**. Set it up (an authenticator app or SMS) when you make the account, or you can't push.

Then work locally: change, add, commit, push

Step 3 — go in and make changes. Open the folder and edit your files:

```
cd my-code                          # step into your project folder
# ...write madlibs.py, guessing_game.py...
```

Step 4 — save a snapshot, then upload it.

```
git add .                           # gather up all your changes
git commit -m "My first Python programs"    # save a snapshot
git push                            # upload it to GitHub
```

Repeat steps 3–4 every time you get something working.