Intro to Coding

Class 4

Booleans Review

- A boolean has 2 possible values: True and False
- Booleans are needed for conditional statements to function

If

- If requires a **boolean** after the keyword
- If the boolean has a **true** value, the code in that **block** (marked by the colon and the indented lines) will be executed
- If the boolean has a **false** value, the code is **skipped**

```
print("Roller Coaster Height Check")
height = int(input("Enter your height in inches: "))
if height < 40:
    print("You cannot ride the roller coaster")
```

Code inside the block

In this case, `height < 40` will return a boolean value. The `if` statement is checking if `height < 40` is **true**. If the boolean has a **true** value, the print statement in the block **will be run**.

Elif

- Elif is short for “else if”
- Elif requires for there to be an if block above it, or else there will be an error
- If the `if` block **above** the `elif` condition is skipped (because the boolean value was false), the program will check the boolean value after the `elif` keyword
- Similar to `if`, the code in the `elif` block will only run if the boolean has a true value

```
if height < 40:
    print("You cannot ride the roller coaster")
elif height < 45:
    print("You must have a guardian accompany you on the ride")
```

Else

- This code block **only** executes when all previous conditional blocks **do not execute** (their boolean values were **false**)
- Thus, this keyword requires that there is an `if`/`elif` block above it
- Similar to “if some action does not happen, I will do something **else**”
- The `else` block does **not** check for a boolean

```
if height < 40:
    print("You cannot ride the roller coaster")
elif height < 45:
    print("You must have a guardian accompany you on the ride")
else:
    print("You may not go on the ride")
```

Conditionals Activity

Write a program that asks the user for their age, and categorizes them in the respective age categories:

Under 18 - Minor

18 to 59 (inclusive) - Adult

60 and above - Senior

```
Enter your age: 17
You are a minor
```

```
Enter your age: 18
You are an adult
```

```
Enter your age: 60
You are a senior
```

Solution

```
age = int(input("Enter your age: "))

if age < 18:
    print("You are a minor")
elif age >= 60:
    print("You are a senior")
else:
    print("You are an adult")
```

Logical Operators

These are additional keywords that can be used in if statements

Common logical operators:

- **and** - evaluates to **True** only if **both** conditions are **True**, false otherwise
  - (i.e. True **and** True will evaluate to True)
  - “If a person is **16 years or older** and **has a license**, then they may drive”
  - **and** takes precedence over everything else (just like how brackets take precedence in the order of operations in math)
- **or** - evaluates to **True** if **at least one** condition is **True**, false otherwise
  - (i.e. True **or** False will evaluate to True)
  - “If a person **has a coupon** or **pays in cash**, then they will receive a discount”

Logical Operators

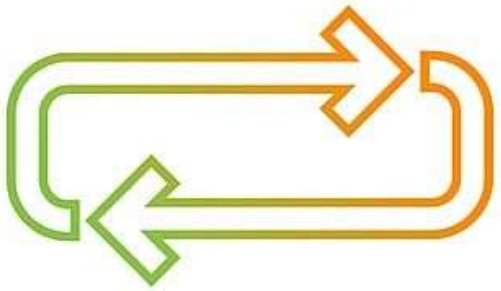
- **not** - evaluates to **True** if the condition is **False**
  - (i.e. **not False** evaluates to **True**)
  - “If the lights are **not off**, then they are **on**”
- **is** and **is not** - another way of checking boolean values
  - (i.e. bool_name **is** True is the same as bool_name == **True**, bool_name **is not** True is the same as bool_name != **True**)
- **in** and **not in** - used to check if a list contains a certain value
  - (i.e. 1 **in** num_list will return **True** if 1 is **present**, and **False** if 1 is **absent**)
- Like mathematical operations, parentheses () can be used to prioritize operators
  - `(True or False) and False` causes the **or** operation to take precedence, resulting in an answer of **False**

Logical Operators Activity

We will put some logical expressions up on screen, and it **is your job to figure out what they evaluate to**

For Loops

- **For loops** are one of the 2 ways to **loop** in Python
- **Loop:** a block of code that is continually repeated until a certain condition is met
- There are 2 different types of for loops: range-based for loops and iterator-based for loops



For Loops

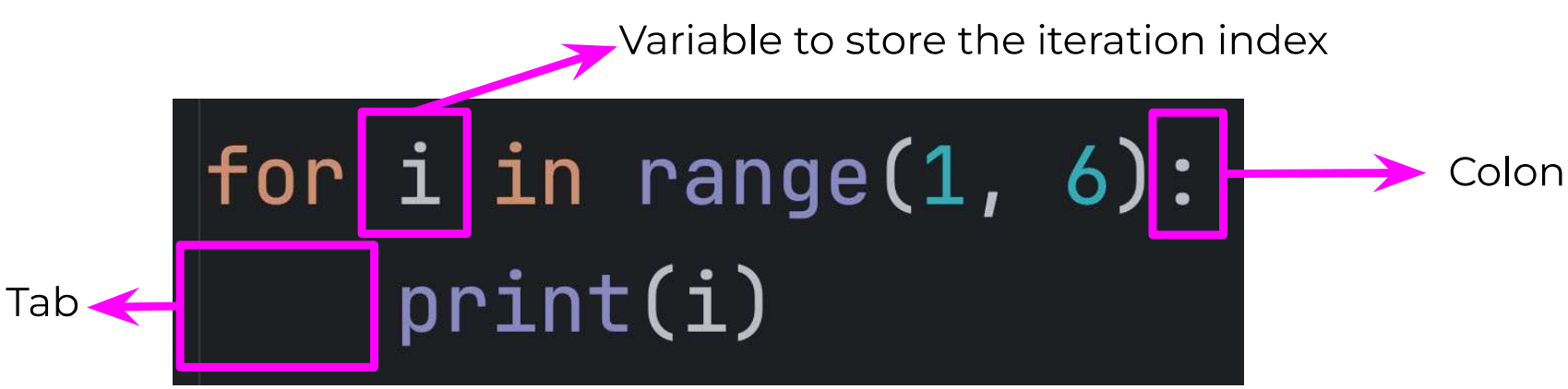
- Range-based with one parameter
- This includes one number in the parentheses
- This is the simplest form, used when you want to repeat an action a certain number of times
- In these types, the iteration index always starts from **0** and ends at **N - 1**, where **N** is the number in the parentheses

```
for i in range(5):
    print("CodeUp")
```

```
CodeUp
CodeUp
CodeUp
CodeUp
CodeUp
```

For Loops

- Range-based with two parameters



Output:

```
1
2
3
4
5
```

- First number is inclusive, second is exclusive!
- This is more commonly used when the start index isn't **0**

For Loops

- Range-based with three parameters

```
for i in range(5, 0, -1):
    print(i)
```

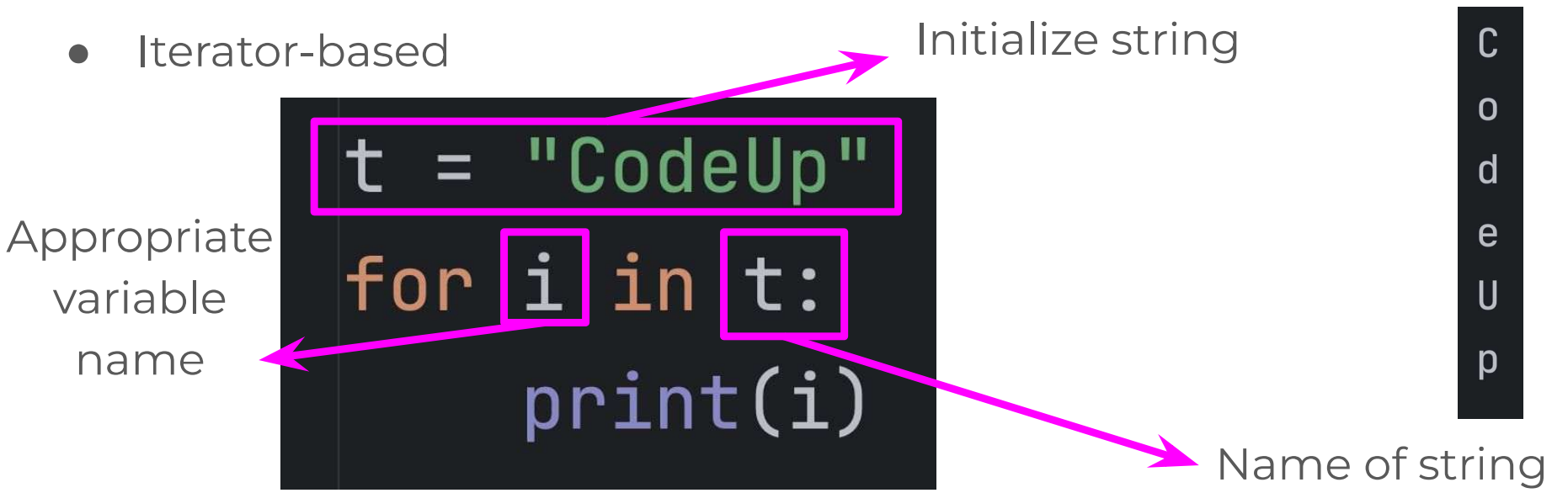
Output:

```
5
4
3
2
1
```

- Adding a **third** parameter will change the **step** of the loop. In this example, the loop will count **down by 1.** The third parameter can be any integer!

For Loops

- Iterator-based



- This will print every character in the string

For Loops

- Iteration-based
- **Does NOT store index value!** If you need to remember the indices, use a **range**:

```
t = "CodeUp"
for i in range(len(t)):
    print(f"{i}: {t[i]}")
```

Output

```
0: C
1: o
2: d
3: e
4: U
5: p
```

- Variables can be used to determine the range!
- If the second number in the range is greater than the length of the list, you will receive an **error**.

Guided Activity: Random DNA Generator

The program takes in input from the user, and generates a DNA sequence of that length. The sequence must be stored in a string for later use.

```
import random

n = int(input("Enter a number: "))
dna = ""
for i in range(n):
    dna += random.choice(["G", "C", "A", "T"])
print(dna)
```

```
Enter a number: 10
GGCCCGTGAG
```

Individual Activity: Random DNA Generator

DNA is double-stranded, but the program only generated one strand. Using the code you have just written, add to it by generating the other strand of DNA.

DNA always follows these rules: A and T must be bonded together, C and G must be bonded together

```
Enter a number: 20
GGTAAGCAACTGCCGTGCCC
CCATTCGTTGACGGCACGGG
```

```
Enter a number: 10
GGTCGGCTTC
CCAGCCGAAG
```

Full Solution

This code generates the other strand

```python
import random

n = int(input("Enter a number: "))
dna = ""
for i in range(n):
    dna += random.choice(["G", "C", "A", "T"])
print(dna)

for i in dna:
    if i == "G":
        print("C", end="")
    if i == "C":
        print("G", end="")
    if i == "A":
        print("T", end="")
    if i == "T":
        print("A", end="")
```

For Loops Activity

Take in 2 lines as input, a string and a character

Find the number of times the character appears in the word

You **cannot** use the `string.find()` function

```
Enter a word: balloon
Enter a character: o
2 result(s) found
```

```
Enter a word: apple
Enter a character: e
1 result(s) found
```

```
Enter a word: dog
Enter a character: a
0 result(s) found
```

Solution

This code simulates what happens in the `string.find()` function

```
s = input("Enter a word: ")
c = input("Enter a character: ")
count = 0
for i in s:
    if i == c:
        count += 1
print(f"{count} result(s) found")
```

While Loop

- Unlike for loops, while loops will **continue looping** while the specified condition is **True**
- **After each** iteration is complete, Python will check if the condition is **True**. If the condition is **True**, then the loop will **continue running**

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

```
i = 1
s = ""
while s != "quit" and i <= 5:
    s = input("> ")
    i += 1
```

While Loops Activity

Write a program that continuously rolls a standard die (numbered from 1 to 6, inclusive) each time the user presses the **enter** key. The user starts with 0 points. The user earns 1 point for each roll that **is not** a 1. The game ends in a **loss** if the user rolls a 1, and ends in a **win** if the user has reached 5 points. The program should print the user's final score, along with if they won or not.

```
Press the enter key to roll:
You rolled a 2
Press the enter key to roll:
You rolled a 6
Press the enter key to roll:
You rolled a 1
Your final score is: 3
You lost!
```

```
Press the enter key to roll:
You rolled a 1
Your final score is: 0
You lost!
```

```
Press the enter key to roll:
You rolled a 2
Press the enter key to roll:
You rolled a 3
Press the enter key to roll:
You rolled a 6
Press the enter key to roll:
You rolled a 4
Press the enter key to roll:
You rolled a 5
Your final score is: 5
You won!
```

Solution

The game continues on two conditions: a 1 hasn't been rolled yet, and there have been less than 5 rolls made. An if statement is used to check if the score should be increased or not.

```
import random

num = 0
sco = 0
while num != 1 and sco < 5:
    input("Press the enter key to roll: ")
    num = random.randint(1, 6)
    print(f"You rolled a {num}")
    if num != 1:
        sco += 1

print(f"Your final score is: {sco}")
if sco == 5:
    print("You won!")
else:
    print("You lost!")
```

Nested Loops

- A nested loop is a loop inside of a loop
- The inner loop will be executed every time the outer loop executes, in general
- Useful for iterating with more complex information, creating more complexity

```
for i in range(5):
    for j in range(10):
```

Nested Loops Activity

Write a program that takes in an integer input **N** and a string. The program will loop through the string, and print each character **N** times

Make sure that all your output is on the **same line**

```
Enter a number: 2
Enter a word: apple
aapppplllee
```

```
Enter a number: 0
Enter a word: CodeUp
```

Solution

Remember to use the end statement to print everything on one line

```
n = int(input("Enter a number: "))
s = input("Enter a word: ")
for i in s:
    for j in range(n):
        print(i, end="")
```

Loop Keywords

- The `break` keyword provides a way to exit a loop early
- The `continue` keyword provides a way to skip the rest of the code in an iteration
- Keywords work for both `for` and `while` loops

```
for i in range(5):
    a = int(input())
    if a > 10:
        break
```

```
for i in range(5):
    a = int(input())
    if a > 10:
        continue
    print(2 * a)
    print(a + 2)
```

Keyword Activity

Write a program that calculates the sum of all **positive even** numbers inputted by the user. The program should stop when the user inputs **-1**, and should **ignore odd** and **negative** numbers **less than -1**.

Your code must include **break** and **continue**.

```
Enter a number: 1
Enter a number: 4
Enter a number: 4
Enter a number: 3
Enter a number: 2
Enter a number: 3
Enter a number: -1
The sum is: 10
```

```
Enter a number: -2
Enter a number: -2
Enter a number: -2
Enter a number: -2
Enter a number: -1
The sum is: 0
```

Solution

```python
s = 0
while True:
    num = int(input("Enter a number: "))
    if num == -1:
        break
    elif num % 2 == 1 or num < -1:
        continue
    s += num
print(f"The sum is: {s}")
```

Error Handling

- Error handling is a method to detect possible errors before they occur, and prevent our programs from terminating.
- For example, if we wanted to receive an input as an integer, we can perform a check to see whether or not we receive an integer or not.
    - This code checks whether the input is an integer, and prints an error message if not

```
number = input("Integer input: ")
try:
    val = int(number)
except ValueError:
    print("InvalidInputError")
```

Error Handling Activity

Create a program that divides number A by number B but can circumvent possible errors

Hint: there is a built-in Python error known as ZeroDivisionError

How would you find the error name that occurs when you don't enter an integer?

```
Enter number A: 3
Enter number B: apple
You did not enter an integer
```

```
Enter number A: 100
Enter number B: 0
Cannot divide by 0
```

Solution

The first `try` statement will attempt to get input from the user. If the user doesn't input an integer, the `except` part will execute.

The second `try` statement will attempt to divide the numbers. If **b** is **0**, it the `except` part will execute.

```
try:
    a = int(input("Enter number A: "))
    b = int(input("Enter number B: "))
    try:
        print(a / b)
    except ZeroDivisionError:
        print("Cannot divide by 0")
except ValueError:
    print("You did not enter an integer")
```

Homework: Guessing Game

Anderson wants to guess the number of dollars that Oscar has in the shortest amount of tries possible. Oscar has between 1 and 100 dollars, inclusive.

Oscar has a random number of dollars, d, and he can tell Anderson if he has more or less dollars than the number Anderson guessed.

Your code should randomly select the number of dollars that Oscar has. You will be acting as Anderson and guessing the number of dollars inside the terminal.

Bonus: can you find a strategy to find the number using at most 7 tries every time?

Sample Interaction

```
Guess how much money Oscar has: 75
Guess higher!
Guess how much money Oscar has: 85
Guess higher!
Guess how much money Oscar has: 95
Guess lower!
Guess how much money Oscar has: 90
Guess lower!
Guess how much money Oscar has: 89
You guessed the correct amount of money in 5 guesses!
```