AI Boot Camp

Programming Basics

Module 2 Day 1

Class Objectives

By the end of class, you will be able to:

1. Use pseudocode to map out a problem.
2. Understand how to use code comments.
3. Understand Python syntax.
4. Create variables.
5. Perform calculations with operators.
6. Print text to the screen with f-strings.
7. Understand basic data types and how to convert them with type casting.

Computational Thinking

Computational thinking **is the process of understanding a complex problem by breaking that problem into smaller parts, and then developing possible solutions that can be clearly presented in a way that computers and/or humans can understand.**

This framework for tackling problems will help you break out foundational concepts as you learn how to code, and will continue to serve you as you encounter increasingly complex problems in this course and in your career as a developer!

The four cornerstones of computational thinking are:

1. Decomposition
2. Pattern recognition
3. Abstraction
4. Testing and debugging algorithms

Instructor **Demonstration**

Introduction to Pseudocoding

Pseudocode: A plain language description of what our code will do.

Let’s explore this by writing an algorithm for putting away dishes.

We will demonstrate computational thinking with pseudocode in code snippets. A code snippet is a portion of code. We will use Python syntax to pseudocode our process, just like we would if we were working in a Python file.

Computational Thinking and Pseudocoding

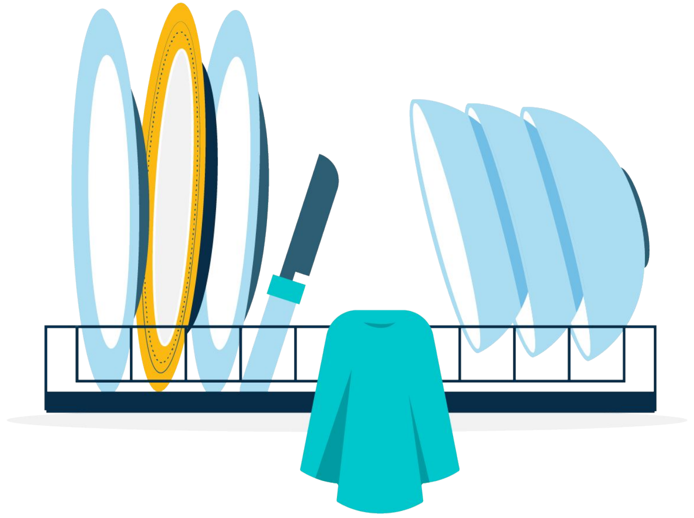
Putting away the dishes the computational way

User story:

- As a member of a household, I want to use computational thinking to develop an algorithm for putting away the dishes.

Acceptance criteria:

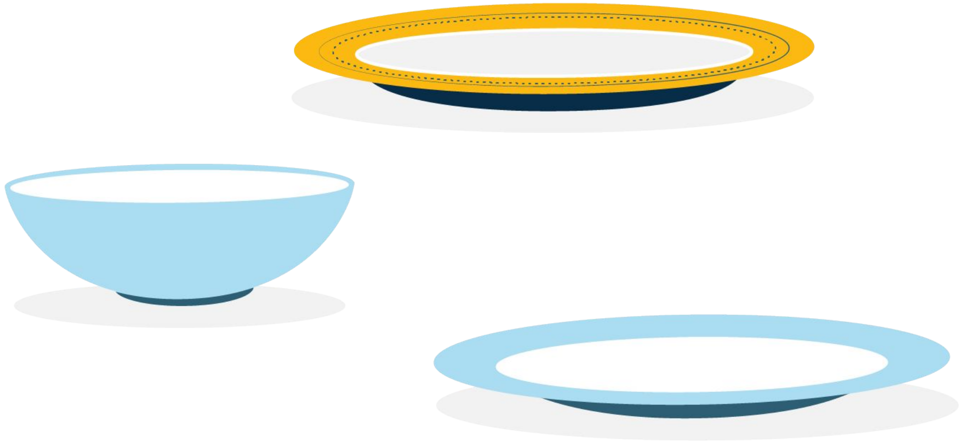
- It is done when I have read through the Code Demo and can apply the four cornerstones of computational thinking (decomposition, pattern recognition, abstraction, and algorithms) to a regularly repeated task.
- It is done when I have identified a task that I complete regularly.
- It is done when I have used pseudocoding to apply the four cornerstones of computational thinking to the daily task.



Decomposition

Decomposition means breaking down a problem into smaller tasks.

Since the problem that we are working on is putting away the dishes, let's take a look at the following image to get an idea of what kind of dishes we are putting away.



Here we see that there are three types of dishes that we need to put away: plain plates, fancy plates, and bowls.

When we pseudocode these subtasks, it might look like the following:

```
# Tasks:
# Stack plain plates
# Stack fancy plates
# Stack plain bowls
```

Pattern Recognition

Think about how we've solved these subtasks previously and try finding any patterns that might help us to solve this particular problem.

The following image shows that our cupboard space is limited and we only have two shelves to work with:

1. In the past, we see that we have first stacked the fancy plates on the top shelf where they are protected.
2. Then we stacked the plain plates on the bottom shelf; and
3. Stacked the plain bowls on top of the plain plates.

Now that we've revisited patterns that we've used to solve this problem in the past, let's pseudocode it:

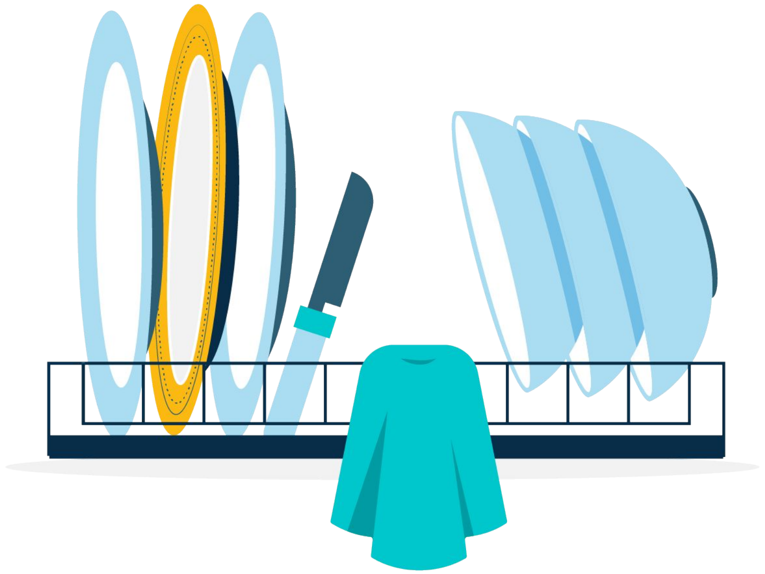
```
# Pattern recognition:
# Plain plates go on the bottom shelf
# Fancy plates go on the top shelf
# Bowls go on the top of plain plates
```

Abstraction

In this step, we will make sure that we are only focusing on relevant information and that we are disregarding details that won't help us solve this problem. Let's take a look at the image of our drying rack again.

We can see that there is a towel on the drying rack that we used to dry the dishes and a knife. Neither of these details will help us solve the problem of putting away the dishes, so let's add some pseudocode to remind us that these details are irrelevant to our solution.

```
# Abstraction:
# Ignore knife
# Ignore towel
```



Sequencing

Now that we've broken down our problem into manageable parts, identified patterns, and eliminated irrelevant information, we can write and test our algorithm.

An algorithm is essentially a sequence of steps or rules that we can use to solve our problem. In our case, it will be the sequence of events that will occur in order for us to complete the task of putting away the dishes.

```
# Sequence:

# 1. Event 1: Put away plates:

# 2. Event 2: Put away bowls:

# Sequence:
# 1. Event 1: Put away plates:
#    Conditional: If (a plate is fancy)
#                   stack it on the top shelf
#                 else
#                   stack it on the bottom shelf
# 2. Event 2: Put away bowls:
```

Debugging

The previous conditional statement would probably work at first, but it is important that we try to think of edge cases, such as exceptions or unusual scenarios, that might occur so that they are covered by our conditional statement.

What if we stumble across a plate that is chipped or broken? Let's adjust our conditional statement so that it includes additional conditions.

Changing our original algorithm after finding a mistake or a potential mistake in our pseudocode is called debugging. We will cover debugging in greater depth in the future. For now, just know that this is an important part of a developer's job!

```
# Sequence:
# 1. Event 1: Put away plates:
#    Conditional: If (a plate is fancy)
#                        stack it on the top shelf
#                    else
#                        stack it on the bottom shelf
#   Debug: A plate has a crack in it - what do we do?
#   Conditional: If (a plate is fancy)
#                        put it on the top shelf
#                  else if (a plate is plain)
#                        put it on the bottom shelf
#                  else if (a plate is cracked)
#                        put it in the trash
#
```

Sequencing our second event

In this case, we only have one type of bowl, so we don't have to worry about making decisions based on certain conditions.

We know that they are all going to be stacked in the same place. Instead of a conditional statement, we can use something called a `for` **loop**, or an action that we will repeat over and over again until a condition is met.

When we pseudocode our loop, it might look something like this.

Here, we have created a `for` loop that directs us to stack each bowl on the bottom shelf on top of the plates.

```
# Sequence:
# 1. Event 1: Put away plates:
#    Conditional: If (a plate is fancy)
#                    stack it on the top shelf
#                  else
#                    stack it on the bottom shelf
#   Debug: A fancy plate has a crack in it - what do we do?
#   Conditional: If (a plate is fancy)
#                    put it on the top shelf
#                 else if (a plate is plain)
#                    put it on the bottom shelf
#                 else if (a plate is cracked)
#                   put it in the trash
# 2. Event 2: Put away bowls:
#    Loop: for (each bowl on the dish rack)
#    stack it on the bottom shelf on top of the plain plates
```

Take a look at the whole thing

```
# Algorithm: Put away clean dishes

# Tasks:
# Stack plain plates
# Stack fancy plates
# Stack plain bowls

# Pattern recognition:
# Plain plates go on the bottom shelf
# Fancy plates go on the top shelf
# Bowls go on the top of plain plates

# Abstraction:
# Ignore knife
# Ignore towel

# Sequence:
# 1. Event 1: Put away plates:
#    Conditional: If (a plate is fancy)
#                   stack it on the top shelf
#                 else
#                   stack it on the bottom shelf
#    Debug: A fancy plate has a crack in it - what do we do?
#    Conditional: If (a plate is fancy) {
#                   put it on the top shelf
#                 else if (a plate is plain)
#                   put it on the bottom shelf
#                  else if (a plate is cracked)
#                    put it in the trash
# 2. Event 2: Put away bowls:
#    Loop: for (each bowl on the dish rack)
#                stack it
```

Fixing the problem:

```
if (plate is cracked)
  put in trash
else if (plate is fancy)
  put on top shelf
else if (plate is plain)
   put on bottom shelf
```

Activity:

Pseudocoding Practice

Write pseudocode for the scenario of sorting and washing laundry.

Suggested Time:

10 Minutes

Time’s up!

Let’s review

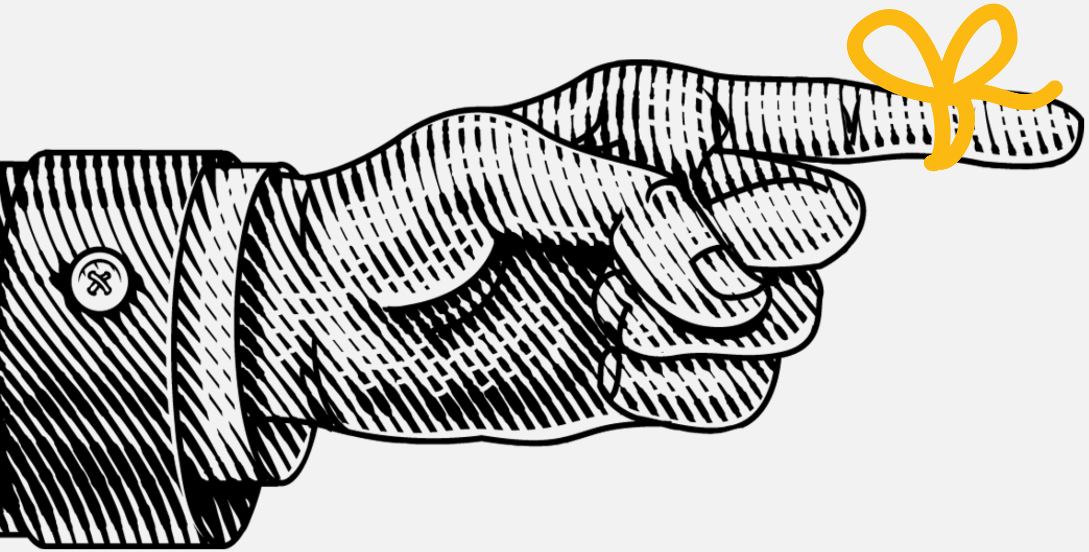
Questions?

Instructor **Demonstration**

Introduction to Python

Writing Comments

Remember the hashes (#) we used in the pseudocode practice?
Those were comments the whole time!



Remember

- No comments are better than bad comments.
- Keep comments up to date when code changes.
- Use complete, clear sentences.
- Prioritize block comments.
- Use inline comments sparingly and only where necessary to clarify obscure or ambiguous code.

Data Types

3
113
-3564
2019

73.81
366.34
-0.13454

“Hello World”
“Fido”
“Paris”

True
False

Integers

- Integers are positive or negative whole numbers used to perform mathematical calculations.
- When typing integer values greater than 999, a thousands separator, or comma, should not be used.

3

113

2019

-3564

Floating-point Numbers

- Like integers, floating-point decimal numbers are used to perform mathematical calculations. Floating-point decimal numbers specify numbers that have a decimal point, like 73.81.

73.81

366.34

-0.13454

Strings

- String variables can be text or numbers wrapped by either single or double quotes, or delimiters.
  - A delimiter is any nonalphabetical character used to specify the boundary between plain text or other numbers, like the single or double quotes, or opening and closing parentheses.
- You may come across empty strings, which contain no text or numbers between the delimiters. A string that is empty is written as `''`, or `""`. Because there are opening and closing single or double quotes, the data type is still a string.

“Hello World”

“Fido”

“Paris”

Booleans

- Boolean data types can have one of two values: true or false.
- To determine the data type of Boolean values, `True` or `False` must be capitalized when written in code.

False

True

Variables are data items with changeable values used for any number of processes or calculations in your code.

You can think of them as containers for the values you will be processing as a part of solving your computing problem.

Creating a variable

Creating is declaring a name

- This name is how you'll refer to the variable throughout the remainder of the Python program.

Maintain good naming conventions for maximum readability

- Be descriptive
  - Instead of `score_a` and `score_b` use `home_team_score` and `away_team_score`
- Variable names should always start with a lowercase letter, and words should be separated with underscores instead of spaces.

Assign data with =

- i.e. `home_team_score = 15, or plate = "fancy"`
- You can also assign variables to variables:
  - `home_team_final_score = home_team_score`

print()

We can use the **print()** function to output our data:

Code

```
profession = "software engineer"

years = 20

hourly_wage = 72.12

expert_status = True

print("Nakia is a professional", profession)

print("They have been working for", years,
"years")

print("Their hourly wage is", hourly_wage)

print("Expert status:", expert_status)
```

Output

```
Nakia is a professional software engineer

They have been working for 20 years

Their hourly wage is 72.12

Expert status: True
```

Activity:

Variables

Create a simple Python application that uses variables and prints strings out to the console.

Suggested Time:

10 Minutes

Time’s up!

Let’s review

Questions?

Instructor **Demonstration**

Inputs and Prompts

Using inputs and data types

```
# Collect the user's input for "What is your
name?"
user_name = input("What is your name? ")

# Print the data type of user_name
print("user_name is type", type(user_name))

# Collect the user's input for "How old are
you?"
# and convert the string to an integer.
age = int(input("How old are you? "))

# Print the data type of age
print("age is type", type(age))
```

- The `input()` function collects values provided by the user as strings
- The `type()` function is used to determine the data type of a variable
- `int()` converts a string to an integer. This is also known as **type casting**.
  - The same principle holds for all data types as long as the value is of a format that can be stored in the new type

Data Types Summary

Data Type	Python Classification	Type Casting Function
Integers	`<class 'int'>`	`int()`
Float Point Numbers	`<class 'float'>`	`float()`
Strings	`<class 'str'>`	`str()`
Boolean	`<class 'bool'>`	`bool()`

Activity:

Input Order

Practice saving inputs from the command line as variables, using string concatenation, and converting data types.

Suggested Time:

15 Minutes

Time’s up!

Let’s review

Break

15 mins

Instructor **Demonstration**

Perform Calculations with Python

Calculation Functions

Operator	Meaning	Use
+	Adds two numbers.	$x + y$
–	Subtracts one number from another.	$x - y$
*	Multiplies two numbers.	$x * y$
/	Divides one number by another. This always results in a floating-point decimal number.	x / y
%	The "%" is known as the modulus. When used in place of "/", it will divide one number by another and return the remainder of the division.	$x \% y$ (remainder of x/y)
//	Divides one number by another and returns an integer. This is known as floor division.	$x // y$
**	Raises a number to a power.	$x ** y$ (x to the power of y)

Order of Precedence

Remember PEMDAS/BIMDAS/BODMAS/BEDMAS order of precedence applies in Python just as it does in mathematics.

```
5 + 2 * 3
8 // 5 - 3
8 + 22 * 2 - 4
16 - 3 / 2 + 7 - 1
3 ** 3 % 5
5 + 9 * 3 / 2 - 4
```

Will have different results to

```
(5 + 2) * 3
(8 // 5) - 3
8 + (22 * (2 - 4))
16 - 3 / (2 + 7) - 1
3 ** (3 % 5)
5 + (9 * 3 / 2 - 4)
```

Activity:

Basic Calculator

Develop an application that uses calculation operators to help a user plan their vacation.

Suggested Time:
15 Minutes

Time’s up!

Let’s review

Instructor **Demonstration**

Printing Complex Strings

Quotation Marks, Escape Characters, and Multiline Strings

- Strings can be wrapped in either single or double quotes (`'` or `"`).
- \ is an escape character you can use to include a quotation mark in the string without it being taken as a part of the overall function.
- \n can be used to add a new line to a string.
- Triple quotes (`"""`) can be used to print text over multiple lines without using \n.
  - It can also be used for f-strings.

F-String Literals

Makes printing outputs very streamlined and can be used in place of more traditional concatenation. This does not mean concatenation doesn't have its place.

Format:

1. The f-string begins with an `f` followed by a string contained within quotes. (The term f-string comes from the leading "f" character preceding the string literal.)
2. In the f-string, curly braces are used to add variables or expressions to the f-string.

Using f-strings what would once have required the following code:

```
radius = 4
pi = 3.14159265358979323846
area = pi * radius ** 2
print("Area: " + str(area) + "cm")
```

Can now be achieved more efficiently:

```
radius = 4
pi = 3.14159265358979323846
print(f"Area: {pi * radius ** 2}cm\n")
```

String Manipulation

- `upper()` converts a string to UPPERCASE
- `lower()` converts a string to lowercase
- `title()` converts a string to Title Case
- The multiplication asterisk, *, will allow you to print a string multiple times:
    - `print(string_variable * 5)` will print the string 5 times.

Multiline f-strings and formatting numbers

Multiline f-string

```
print(f"Text {variable}. More text\n"
      + f"Text: {calculation}\n"
```

- Use the f at the beginning of each f-string.
- Concatenate the string at the beginning of each line
- End the f-string with a new line character (\n)

Formatting numbers

```
f'{value:{width},.{precision}}'
```

- Width specifies the number of characters used to display the value.
- Precision indicates the number of decimal places to format the value.
- The comma should only be included when a thousands separator is required.

Examples

\n: `print("Name: Michelle Yeoh\nDate of birth: August 6, 1962\nBirth place: Ipoh, Malaysia")`

Triple quote:

```
text = """

Name: Michelle Yeoh

Date of birth: August 6, 1962

Birth place: Ipoh, Malaysia

"""

print(text)
```

Triple quote f-string: `f"""Some text {a_variable}"""`

String multiplication: `print(string_variable * 5)`

Activity:

Print Presentation

Practice printing more complex strings, along with reinforcing other Python skills from this class.

Suggested Time:

15 Minutes

Time's up!

Let's review

Let's **recap**

Recap

Today you learned about:

1. Using pseudocode to map out a problem.
2. Creating variables in Python.
3. Basic data types and how to convert them with type casting.
4. User inputs and printing to the screen.
5. Performing calculations with operators.
6. Creating, using, and manipulating strings and f-strings.

Next

This week's challenge will involve creating an interactive menu for a food truck for customers to order from, which will print the customer's order and total cost once the order is complete.

Questions?

The End