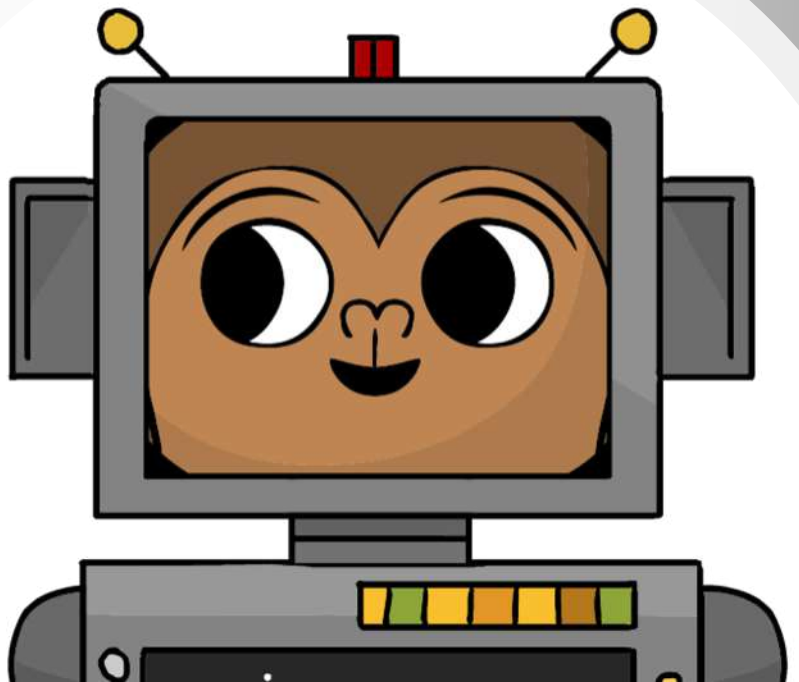
codemonkey
LESSON PLANS

CODING CHATBOTS

Learn Python by Programming Chatbots

Lesson Plans 1-16



CodeMonkey
LESSON PLANS

2345 Yale St., 1st floor
Palo Alto, CA 94306
info@codemonkey.com
www. codemonkey.com

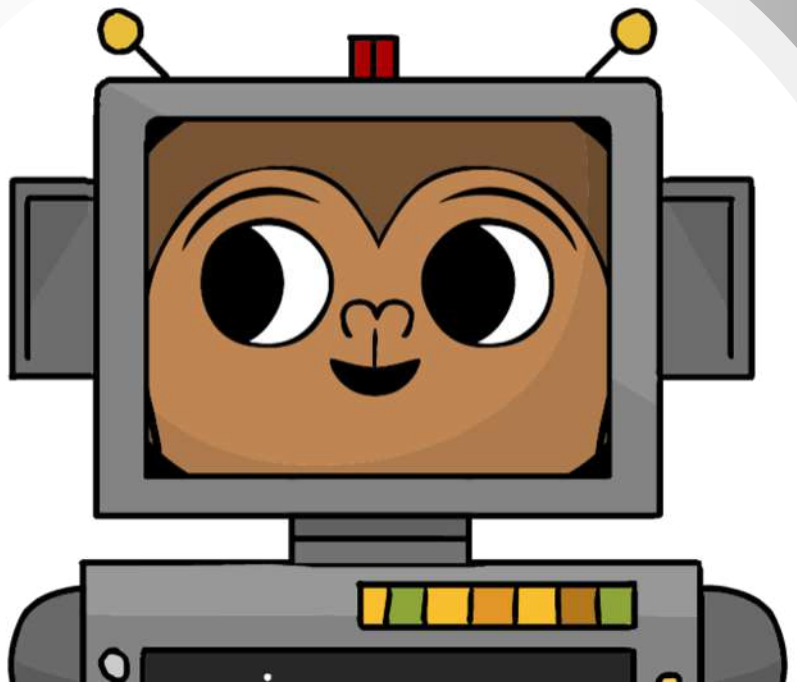


Table of Contents

Introduction

Thank you for choosing CODING CHATBOTS to teach your students how to code in Python. In this course, students will code a chatbot program that will host a popular guessing game where players need to guess the correct letters in a secret word. Students will write the entire Python code by themselves. This course is recommended for **7th-9th graders**.

We recommend that teachers have moderate background in coding to teach this course. For students, previous coding experience is an advantage. The following 16 lesson plans will cover all 74 exercises. Each lesson is made up of 3 parts: an Introduction, Playtime and Debriefing. Each lesson is broken down to 45 minutes, however, some may take longer to complete. If you are rushed for time, please skip to part 2.

At the end of this document, you will find a Glossary that summarizes each coding concept. Please refer to this throughout the course. For information regarding setting up a class, please read A Beginner's Guide to CodeMonkey, located in the Teacher's Resources Menu on your homepage.
Please email us with any questions.

Good Luck!!
The CodeMonkey Team

STANDARDS

*visit course page for detailed-alignment

CSTA K-12 COMPUTER SCIENCE

- KEY STAGE 1
- KEY STAGE 2
- KEY STAGE 3
- KEY STAGE 4

COMMON CORE STATE STANDARDS

- CCSS.MATH.PRACTICE.MP1
- CCSS.MATH.PRACTICE.MP3
- CCSS.MATH.PRACTICE.MP4
- CCSS.MATH.PRACTICE.MP5
- CCSS.MATH.CONTENT.6.NS.C.5
- CCSS.MATH.CONTENT.6.EE.B.6
- CCSS.ELA-Literacy.RST.6-8.3
- CCSS.ELA-Literacy.RST.6-8.7

NATIONAL CURRICULUM IN ENGLAND

- Level 1/K-3/3-6 : Computational Thinking
- Level 1/K-3/3-6: Computing Practice & Programming
- Level 2/6-9: Computational Thinking
- Level 2/6-9: Collaboration
- Level 2/6-9: Computing Practice & Programming
- Level 2/6-9: Computers & Communication Devices
- Level 3A/9-12: Computational Thinking
- Level 3A/9-12: Collaboration
- Level 3B/9-12: Collaboration

Lesson 1 – Send & Receive

In this lesson, students will play a game that will help them prepare for the course ahead. They will also become familiar with the Coding Chatbots environment.

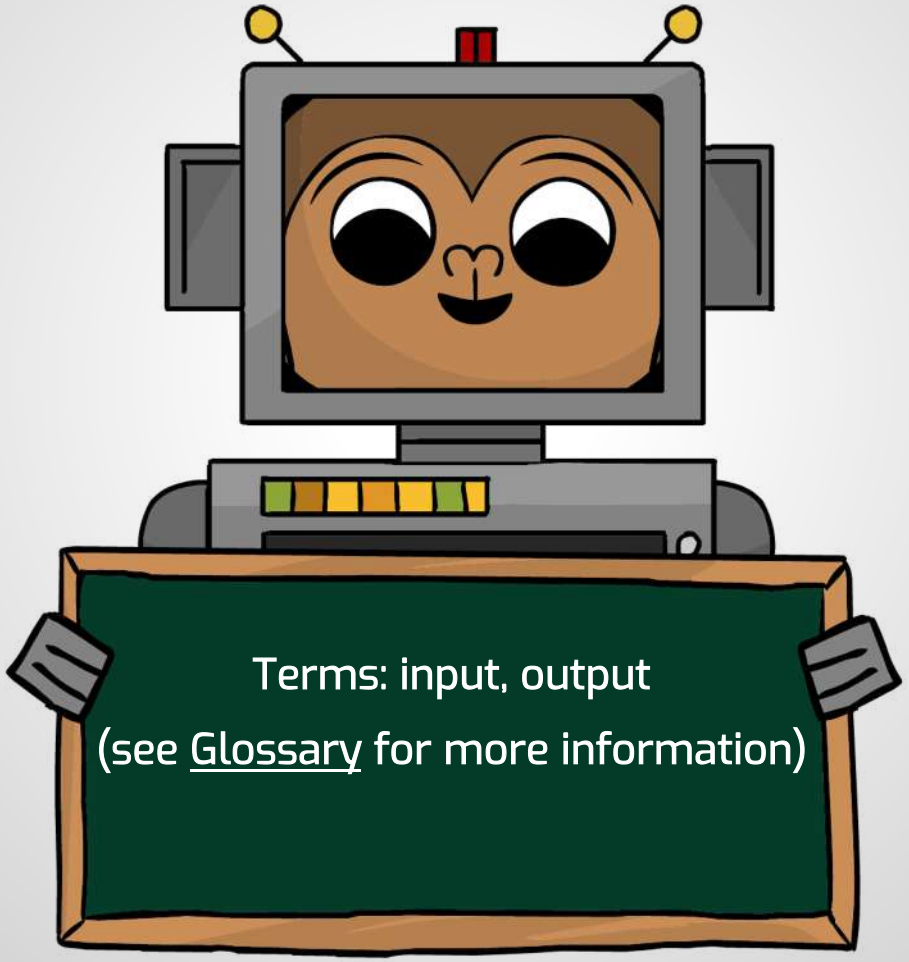
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Write short Python code
- Use output (send_message) and input (read_message)
- Use ‘test’ when they want to run automated tests on the chatbot and ‘chat’ when they want to run their own checks.
- Complete exercises 1-4

Components



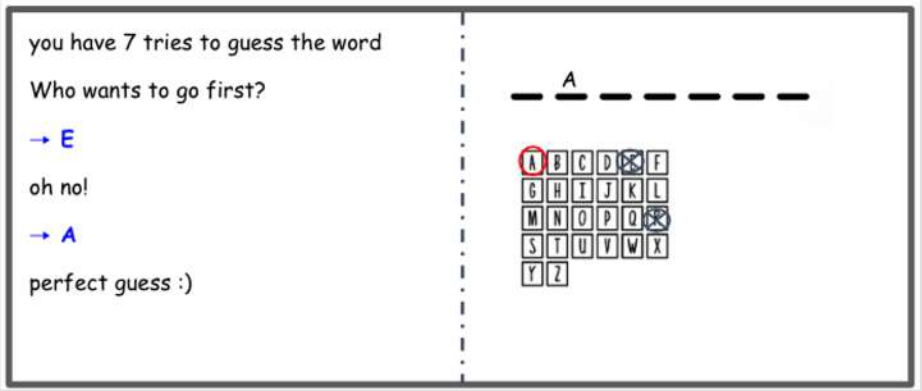
Part 1: 12 Minutes

Introduction

Activity – Play “Snowman”	**5 mins.**

Today you will be playing Hangman, or as we will refer to it ‘Snowman’. This game will focus on aspects of chatbot interface, specifically input and output messaging.

For this activity, you or a student will take on the role of the host and play Snowman with your class. The host should refrain from talking in this activity – only write on the board. **Be sure to write the full interaction on the board.**
The word your students will need to guess is **Hangman**
(feel free to choose another word).
1. Start this task by writing the following on the board:
You have **7 tries** to guess the word -
“ __ __ __ __ __ __ __ __”
Who wants to go first?



Play this game with your class by writing all instructions and guessed letters on the left-side of the board as shown above. If you have a longer class than 45 minutes, you can play this game a couple of times with different students acting as the host.

Part 1: 12 Minutes

Introduction

Discussion	**5 mins.**

Lead a discussion with your class about the game by asking the following questions:

1. What is on the left-side of the board?
   - The transcript of the game. The clear depiction of instructions between the host and students.
2. Do you see a difference in the text?
   - Questions vs. answers format.
3. Who is responsible for the output? (aka the instructions)
   - The host or chatbot
4. Who is responsible for the input? (aka the guesses)
   - The students or players
5. What is the game flow?
   - A word is chosen by the host and the number of letters in the word are marked on the board. Students guess letters one at a time. The host checks to see if the letter is in the word and the guessed letters are added either to the word on the board or in a bank with other wrongly guessed words. The number of guesses left are updated.

Part 1: 12 Minutes

Introduction Cont.

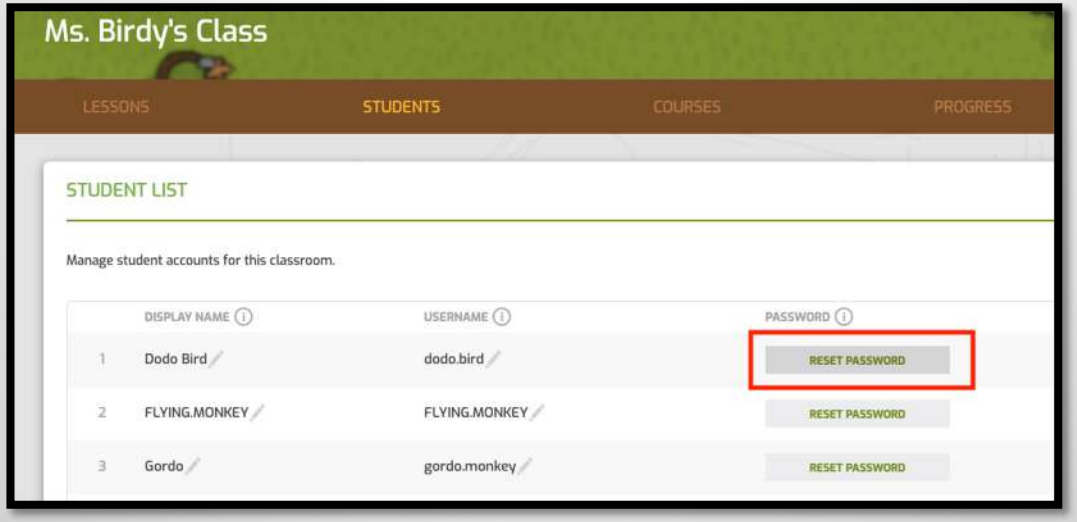
Discussion	**2 mins.**
Summarize the discussion by playing the video below, which will show what students will build at the end of this course. Point out the output messages and input messages between the chatbot and player are similar to the game you just played.	

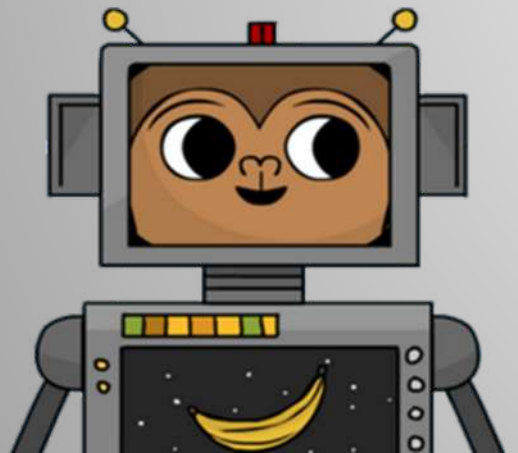
Part 2: 28 Minutes

Playtime

Log-in	1 min.

Instruct your class on how to log into their CodeMonkey accounts and go to the Coding Chatbots activity.
If your students use usernames and passwords to log-in, make sure they store their information where they can easily access them in the future. You can also handout user log-in cards. If a student forgets their password, you can reset it by visiting the classroom dashboard, going to the Students tab, and clicking on reset password.





Part 2: 28 Minutes

Playtime Cont.

Explanation - Interface	9 mins.

1. Open Coding Chatbots
2. Show how the environment is divided into 3 sections: the instructions (left), the Python editor (center) and the chatbot (right).
3. Send message("text") displays messages in the chatbot and the text that you want to send needs to be written inside the quotation marks in order to work. Each exercise in the course includes tasks that need to be completed.
4. After making the changes required in the task, it is completed by clicking on the "check" button.
5. Upon successful completion of an exercise, a green checkmark appears and the next exercise becomes available.
6. If the changes to the code are incorrect, the task is marked as "failed" and a retry is needed.
7. Use the example in exercise #1 to explain how to use the editor, check and chat buttons, and send_message("text").
8. Ask your students to change the text as required in the exercise. Then, they need to hit check and hit chat.
9. ****Make sure the text in the send_message is identical to the text required in the instructions.****
10. The final task in each exercise is to run your chatbot.
11. Ask your students to:
    - Change the text in the editor as required in exercise #1
    - Click CHECK
    - Click CHAT to run their program

Part 2: 28 Minutes

Playtime Cont.

Playtime	**5 mins.**
Have your class complete exercises 1 and 2. Encourage them to go step by step. Make sure they use the same exact text required by the exercise or else it will not pass.	
Explanation	**4 mins.**
Let's learn how to chat - Ask your class what defines a chat/conversation? • Question & Answer • Request & Response In this game, instead of a question and answer to conduct a conversation, we will use send_message("text") and read_message() read_message() allows you to write your responses to questions in the bottom half of the chatbot Present the first instruction of exercise 3. **For more information on read and send message, please refer to the Glossary.**	

Part 2: 28 Minutes

Playtime Cont.

Playtime	**5 mins.**
Have students complete exercise 3. You can use your classroom dashboard to keep track of student progress.	
Explanation	**4 mins.**

In exercise #4 we will learn the difference between TEST and CHAT.
Please explain the following:

- TEST is an internal checking mechanism that causes the messages to be sent. When a read_message is reached in the code, an automatic response is given to the chatbot. The user does not need to enter anything.
  - Use Test mode to check the code's compliance with the game instructions
- CHAT is a Python compiler that causes the messages to be sent. When a read_message is reached in the code, the execution is halted, and the chatbot waits for the user to respond. Once the user responds, the chatbot continues running.
  - Use chat mode to give additional examples that are not included in the game.

Part 3: 5 Minutes

Debriefing

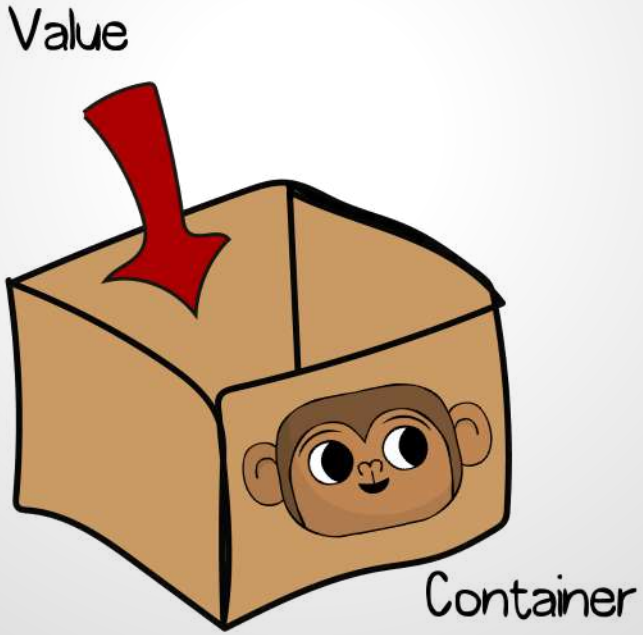
Discussion	**5 mins.**

Ask your class to name something they learned today. Remind them of the following:

1. The Coding Chatbots environment:
   - CHECK
   - TEST
   - CHAT
2. The methods:
   - send_message("text")
   - read_message()
3. Ask what happens if the text in the send_message("") is not identical to the instructions?
   - The task will not pass
4. Ask what happens if no one replies back to a read_message()?
   - The code will not run until a response is entered

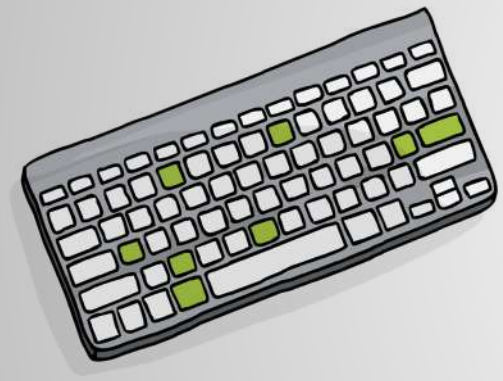
Lesson 2 – Variables as Containers

This lesson focuses on how and when to use variables.



Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

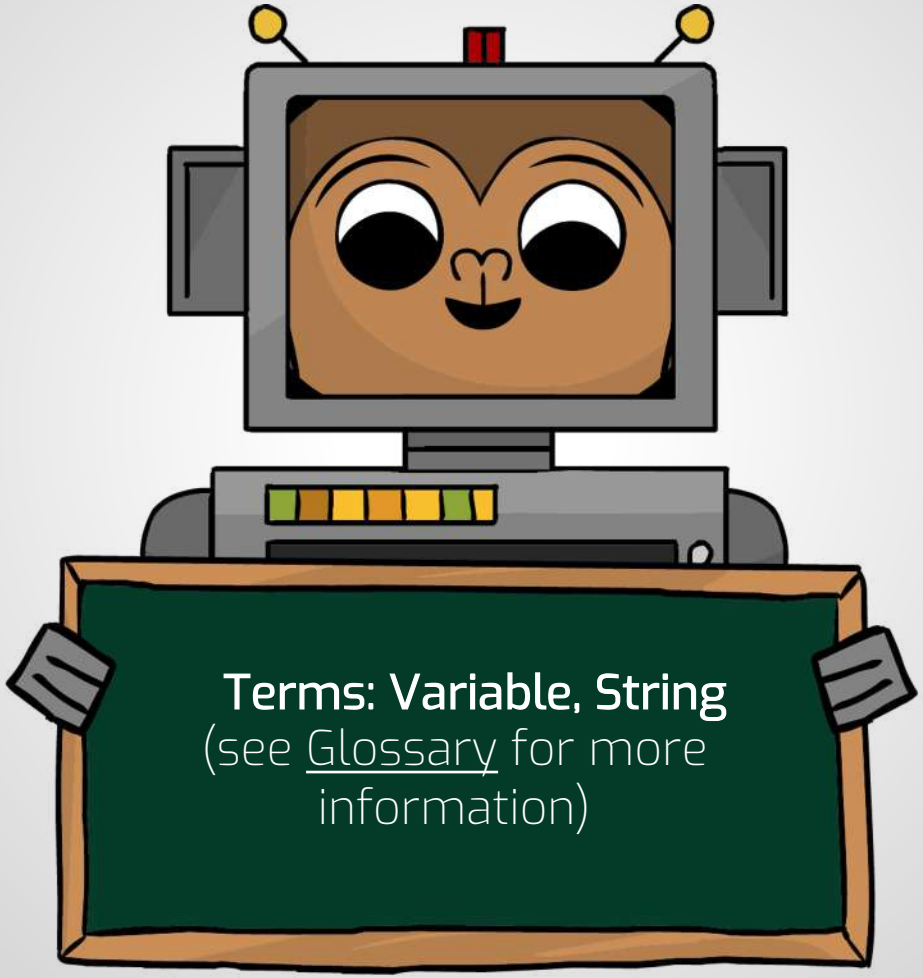
Objectives



Students will:

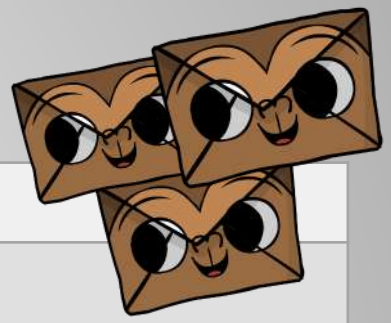
- Create situations that require using variables
- Define and assign variables
- Use variables in the context of send/read methods
- Understand the difference between a variable's name and its value
- Concatenate variables into strings
- Complete exercises 5-8

Components



Part 1: 20 Minutes

Introduction



Activity - Variables	**10 min.**

Today's lesson will start with an interactive activity that will illustrate how variables and containers work.
Please prepare the following in advance:

- 3 envelopes marked A, B & C (envelopes will represent the containers/variables)
- 10 index cards numbered 1-10 and a blank index card if needed later on (these numbers will represent the values)

- Call on 3 volunteers to the front of the room. They will represent the program we are coding.
- Place the 3 envelopes and the index cards on a desk. Explain that the envelopes will play the role of containers and the index cards will represent the values.
- Ask the volunteers to each choose an index card.
- Ask 2 of the volunteers to put their card in an envelope..
- Then, ask the 3rd volunteer to put their card in one of the full envelopes. Ask the class what will happen to the value that already exists? Can we have 2 values in one envelope? Can a container or variable hold two values? (Answer is no, a container or variable can only hold one value at a time)
- The student should takeout the index card that the previous volunteer put and put his or her own index card in the envelope. This is when you emphasize that a variable can only hold a single value at a time. If you took out a value, you cannot recall it.
- Then ask the class, what is in the empty envelope? They may answer zero. The answer is that the envelope's value is nothing, which is not the same as 0.

Part 1: 20 Minutes

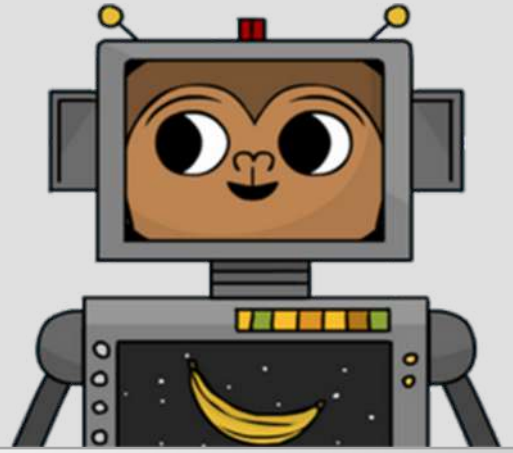
Introduction Cont.

Activity – Variables Cont.	**10 min.**

- Now, we will assign a value to the empty envelope using the index cards in the existing envelopes. For example, if envelope A had the number 8 in it and envelope B had the number 4 in it, we will now assign envelope C the value 12. So take a blank index card, write 12 on it, and place it inside envelope C.
- Write on the board **C = A +B.** Ask how do we read it? What does it mean? Some will say will say C equals A + B, however this is not how we read it in programming. In programming, it is an assignment or placement (When you place the value into the container.) So in programming, it is read as A + B is placed into C. Mention to the class that the container (envelope) must always stay on the left-side of the equation . The value (index card) is always on the right-side. So that is why we can say it is not like an equation since in programming, 3 = C is not the same as C = 3).
- To further illustrate this, change the value of one of the previously full envelopes and ask what happens now? Will the value in the previously empty envelope change as well? The answer is no.

Part 1: 20 Minutes

Introduction Cont.

Discussion	**5 mins.**
Ask the students the following: 1. How would you define a variable? Variables hold values - numbers, text (and more) 2. Does a variable have a name? Yes, the programmer can choose a meaningful_name 3. What happens when you assign a new value to a variable? You destroy anything that was previously there 4. How do you refer to a variable? By it's name *Do not confuse variable names with their values	

Part 1: 20 Minutes

Introduction Cont.

Common Misconceptions about Variables	**5 mins.**

Point out for the following common misconceptions:

- Variables in programming are similar to variables in Math (students get these confused – they are not the same)
- Assignment is like a mathematical equation – both sides are equal and can therefore be switched (No, the right hand value gets assigned to the left hand value so they can not be switched)
- A variable still holds its original value after an assignment (No, after an assignment, the original value gets replaced by the new one)
- A sequence of assignments all happen together at the same time and can happen in any order (False, a sequence happens from top to bottom and order is important)
- After an assignment, any future change to one variable changes the other (false: the connection between 2 variables is not a binding relationship for ever)

Part 2: 20 Minutes

Playtime

Explanation	**7 mins.**

Go to exercise 5 in Coding Chatbots.

Edit the following code:

```
language = "Python"
send_message("I speak")
send_message(language)
```

Before you click CHAT, ask the class what will happen in the chat? Here the player has no role.

Click CHAT

Add the following instructions

```
send_message("What do language you speak")
language=read_message()
send_message(language)
send_message("is important to know")
```

Before you click CHAT, ask the class what is the difference from the first 3 instructions. Here the use of read_message() halts the game until the player responds.

Click CHAT

Part 2: 20 Minutes

Playtime Cont.

Log-in	**1 mins.**
Review log-in instructions here.	
Playtime	**22 mins.**
Ask students to complete exercises 5-8. Encourage them to go step by step. Pay attention to use the exact text required in the exercise.	

Part 3: 5 Minutes
Debriefing

Discussion	**5 mins.**

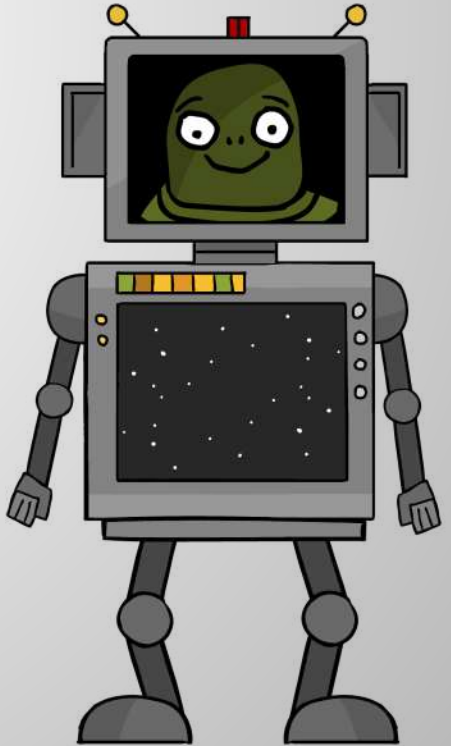
Ask your class to come up with one thing they learned today, leading questions could be:

1. What did we use the envelope for?
   - Stand as variable
2. What type of values can we assign to a variable?
   - Numeric, string, letters
3. What happens when we change the value of the variable?
   - It changes from that moment on, you cannot recall the old value if you didn't save it somewhere else
4. How can we assign a value from the user?
   - read_message()
5. What is the difference between variable name and its value?
   - The name is the way we refer to the variable and it does not change. The value can change
6. What are variables good for?
   - To store data. If you delete it, variables hold all the data we use. And we can go back to this data to use it as we compute.

Lesson 3 – Conditional Instructions

The lesson will start with a short interactive guessing game like in the first class, except this time it will focus on defining conditional statements. The main part of the lesson is aimed at practicing the use of conditional statements including:

- in operator
- if/else



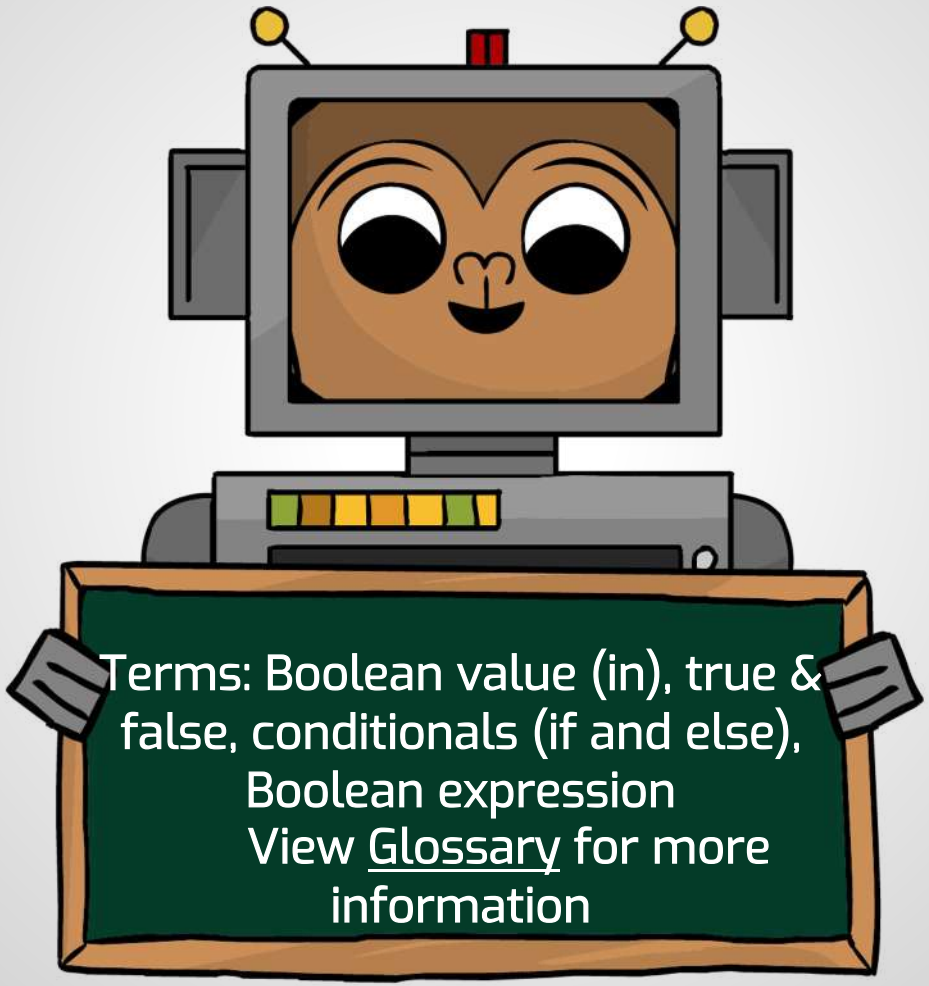
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Learn conditional execution
- Define a situation when a part of the program should run or not
- Know how to use the ‘in’ operator
- Get a glimpse into Boolean values – True and False
- Understand the difference between if and else
- Complete exercises 9-14

Components



Part 1: 10 Minutes

Introduction

Activity – Conditional 'Snowman'	**5 mins.**
Choose a student to play the host and have them choose a word that is 5-7 letters long. The host should draw dashes on the board to match the length of the word. To guess the word, students can only ask true or false statements such as is **the letter X is in the word** and the host can only answer with either true or false. The host will draw parts of a snowman each time students guess wrong. Students will play until the word was guessed or the snowman was drawn. Make sure the host only answers when the question is phrased as defined above. By asking is blank letter in the word , students are practicing a **condition** statement in Python. True or false is the only answer of this statement in Python.	

Part 1: 10 Minutes

Introduction Cont.

Discussion	**5 mins.**

- Ask your students what was unique about the game this time (the students can only ask questions in a certain way like in the programming language python)
  - What was unique about the question they asked -
    - Can a letter be **in** and **not-in** the word at the same time? Can true and false exists at the same? No, a single condition can only either be true or false
  - Discuss the fact that there is only one valid answer
- Present the concept of **true** and **false:**
  - True and False are Boolean values used for making decisions.
  - There are different ways of comparing values - We can test if a character exists in a string or if a number is greater/smaller than another number, if they are equal or if they are different.
  - In each case, the value of expression is true or false.

Part 2: 35 Minutes

Playtime

Explanation – in	**5 mins.**

Write the following on the board:
(Ask students to reply back with **true** or **false)**

- "P" in "Python"
- "on" in "Python"
- "a" in "Python"
- “p” in “Python” → *note that Python is case sensitive*

Introduce the **in** operator for strings, which is used to test whether a value is found within a sequence
Complete exercise 9 with the class
Ask your students how they can use the ‘in’ operator

- They can use it in send_message and read_message (as practiced the first lesson)

Ask your class what the result of running send_message(“Py" in "Python") will be

- They will see true written in the chatbot

Using variables within the ‘in’ operator, present the following:

```
response = read_message() #response will get the user’s response or it can be a symbol
send_message(response in “Python”) #this will check if the letter is in “Python”
```

Part 2: 5 Minutes

Playtime

Explanation – if	**5 mins.**

Conditional checks are important for comparing numeric values

- What will the output be?

```
if 7 > 0:
    send_message("7 is greater than 0")
```

- And in this case -

```
if -7 > 0:
    send_message("7 is greater than 0")
```

- **It is important to emphasize that if the condition is false, then the instructions under the if are not executed at all**
- Pay attention that each if starts at the beginning of the line
- At the end of the ‘ if’ you always need to use ‘:’
- Make sure the instructions within the ‘if’ are indented one tab

Part 2: 5 Minutes

Playtime

Explanation – else	**5 mins.**

How can we define a different set of instructions if the condition is false?

- An **if** statement can be followed by an **else** statement.

```
if "x" in "Python":
        send_message("x is in Python, really??")
else:
        send_message("x is not in Python")
```

- Complete exercise 11
- Pay attention that each else starts at the beginning of the line
- At the end of 'else' you always need to use ':'
- Make sure the instructions within the 'else' are indented one tab

1. Present the following on the board and ask your to write the output of each code
2. Discuss the difference between the code on the left and the code on the right

- `if` condition is **True**
  - code inside **if** is executed
- `if` condition is **False**
  - code inside **else** is executed

```
if "x" in "Python":
    send_message("x is in Python, really??")
else:
    send_message("x is not in Python")
```

```
if "o" in "Python":
    send_message("o is in Python")
send_message("o is not in Python")
```

Part 2: 5 Minutes

Playtime

Log-in	**1 mins.**
View log-in instructions here.	
Playtime	**20 mins.**
Have students to complete exercises 9 through 14 • Ex. 9-11 covers each of the topics presented in the lesson • Ex. 12-14 integrates the new material into the game using variables, send and read instructions. Encourage the students to go step by step. Pay attention to use the exact text as required in the exercise Tip: use your classroom dashboard to keep track of student achievements	

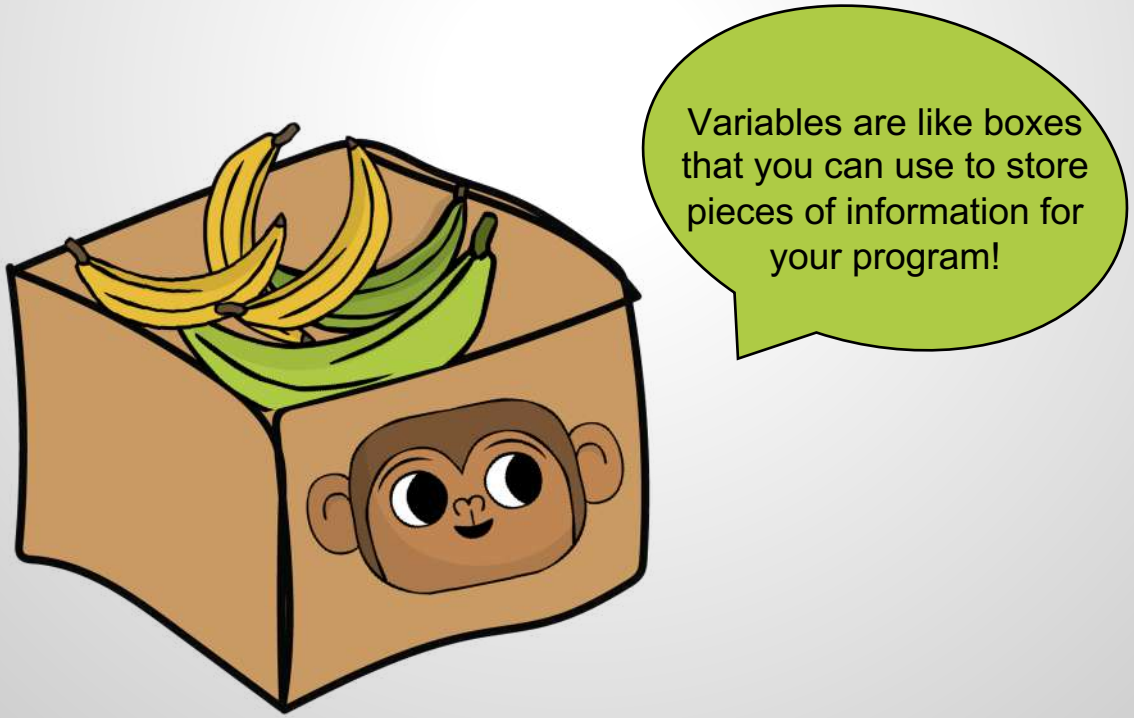
Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**
Ask students to come up with a situation in real life where they use conditions: •Conditions are everywhere even if we don't explicitly notice •A condition can either be True or False, but never both •For example, if it rains tomorrow, we will not have school.	

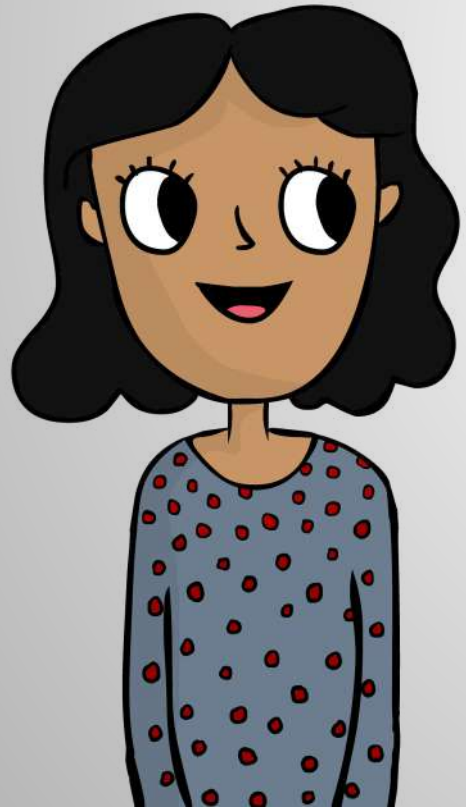
Lesson 4 – Advanced Variables

The main part of the lesson is aimed at practicing the life-cycle of variables - declaring, initializing, assigning and changing variables. In addition, students will practice the fundamental string and arithmetic operators for concatenating strings and adding values using = and +=.



Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

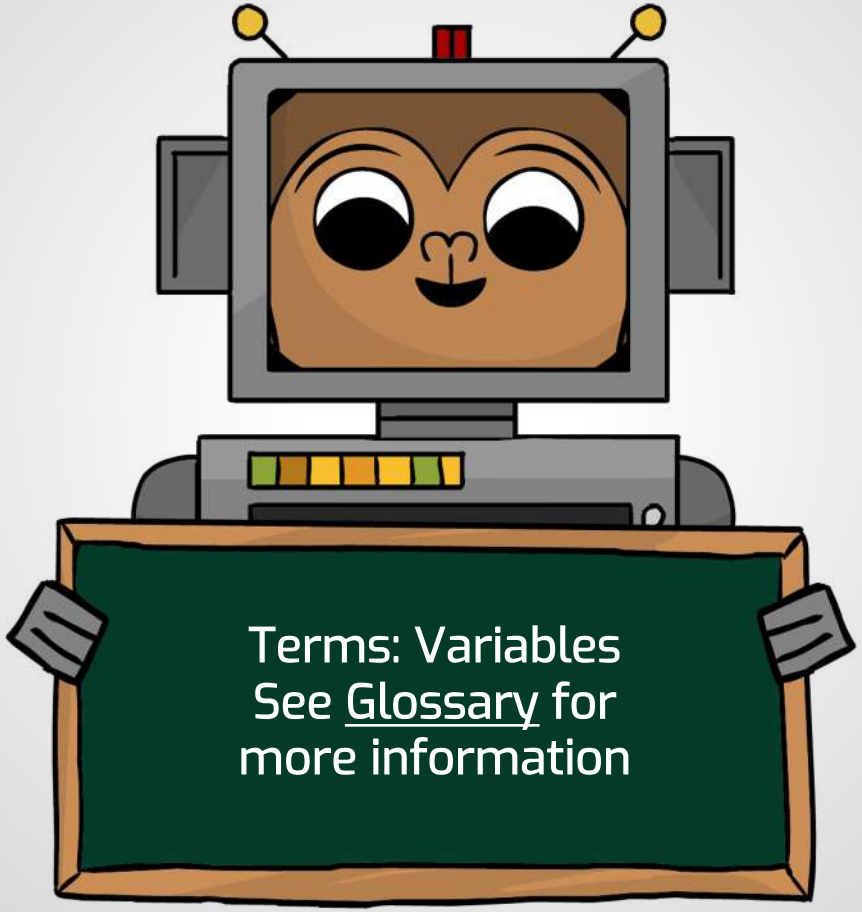
Objectives



Students will:

- Gain a deeper understanding of the concept of variables
- Learn how to initialize and change the the value of a variable
- Have an advanced understanding of variables: get to know the difference between number and sting variables and how to use them in Python
  - Concatenate string
  - Arithmetic operators like /+=/-+
- Complete exercises 15-18

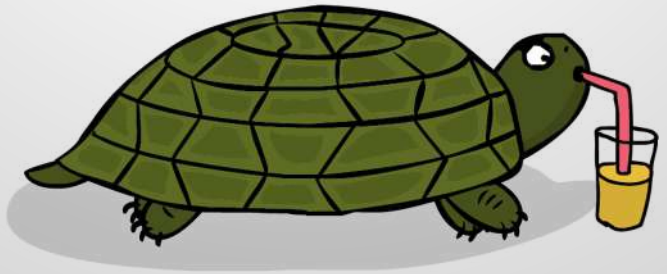
Components



Part 1: 5 Minutes

Introduction

Activity – Cups	**5 min.**
We will demonstrate what swapping values means through the representation of different beverages in cups. In this activity, you need 3 glasses, 2 containing different-colored drinks (i.e. orange juice and water). Ask the class how you can switch between the two liquids? • Answer: use the third empty cup. • This is a metaphor for swapping values contained in variables. • In order to swap variables, we need to use a third temporary variable.	



Part 1: 5 Minutes

Introduction Cont.

Activity - Advanced	**5 mins.**

If the class is familiar with variables swap with using a temp variable, present the following code:
Ask the class what will be the output and go line by line to follow the code and variables values

```
var_1 = 5
var_2 = 7
var_1 = var_2 + var_1
var_2= var_1 - var_2
var_1 = var_1 - var_2
send_message(var_1)
send_message(var_2)
```

Part 2: 35 Minutes

Playtime

Explanation	**7 mins.**
This part correlates to exercise 15. Variable value can change along the program Assigning new value •my_var = "snowman” •my_var = "Python” Assigning value from input •my_var=read_message() Assigning value from input or programmer-defined can be done in any place in the code Once the value has changed the previous value no longer exist	

Part 2: 35 Minutes

Playtime Cont.

Explanation – Concatenate	**5 mins.**

Concatenate: to link strings and characters together in a chain or series

Strings can be changed by adding additional characters to an existing string

The + operator concatenates two strings

Ask the class what will be the value of the variable ‘name’ after each of the following instructions?

```
name = "Pyth"
letter_1 = "o"
letter_2 = "n"
name = name + letter_1
name += letter_2
```

Emphasize the ability to use the long and short versions of the concatenate operator as presented in this example.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Numeric Variable	**5 mins.**

Numeric variables in Python are used to store numbers as integers or floating (rational) numbers.
Variables can hold a single value of a single type at a time

```
my_var_1 = 5 #number
my_var_2 = "5" #string
```

The arithmetic operators in Python are used to perform mathematical operations such as addition, subtraction, multiplication, and division. In this lesson we will focus on addition and subtraction.
What will the output be?

```
my_var = 0 # holds 0
my_var = 3 # holds 3
my_var += 1 # holds 4
my_var -= 2 # holds 2
my_var_1 += 75 # holds 80
my_var_2 += "75" # holds "575”
```

Emphasize the ability to use the long and short versions of the concatenate operator as presented in the example above.

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 mins.**
Review log-in instructions here.	
Playtime	**20 mins.**
Ask students to complete exercises 15 - 18. Encourage them to go step by step. Make sure they use the exact text required in the exercise. Use your classroom's dashboard to keep track of students' achievements.	

Part 3: 5 Minutes

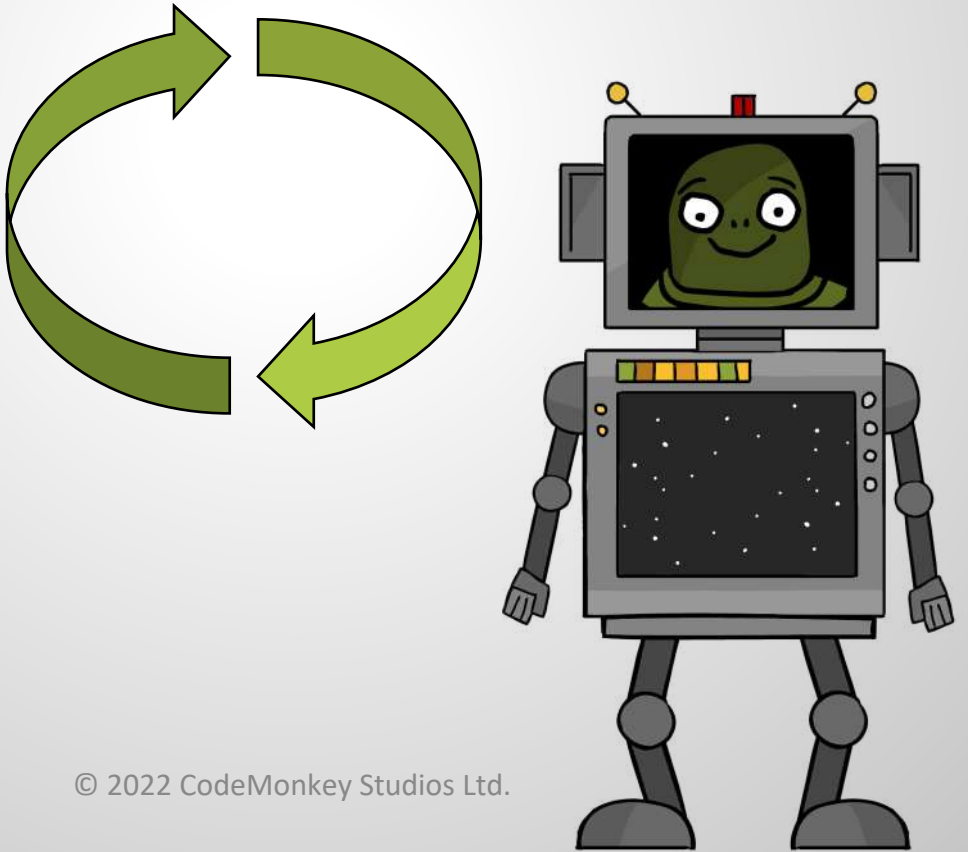
Debriefing

Debriefing - Variables	**5 mins.**

•Variables hold values, the actual data that is being stored in memory
•Variables have names so programmers and the computer know which variable is called on
•Do not confuse variable names with their values
•A variable can only store one value of one type at a time
•When you change the value (or the type) in a variable, you destroy anything that was previously there

Lesson 5 – Repetitive Execution

The first part of this lesson will focus on introducing the students the range() function. Then we will dive into the For loop structure using range().



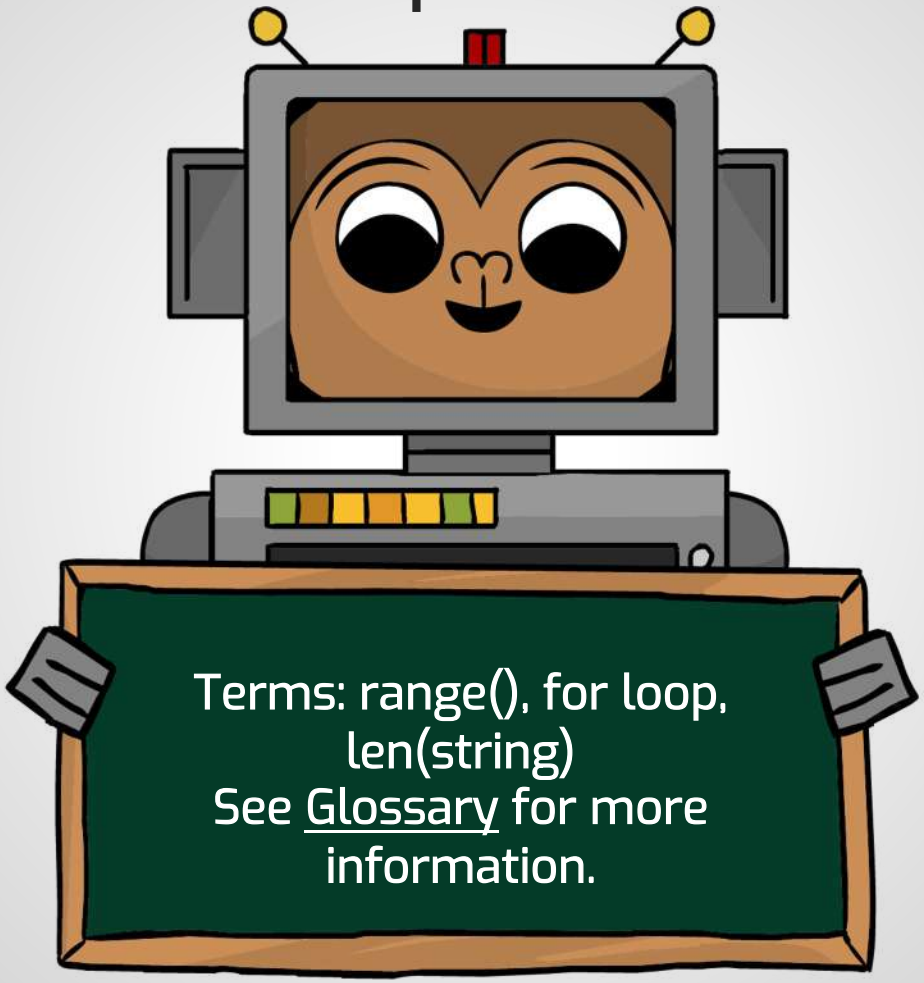
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Understand the importance of loops (that it is more efficient for the programmer than writing repetitive code)
- Identify repetitive code and convert a series of multiple repetitive actions into a loop
- Learn how to use the range() function and For loops
- Learn how to use len(string)
- Complete exercises 19-24

Components



Part 1: 5 Minutes

Introduction

Activity	5 mins.
• Start with a recap of the previous lesson by presenting the code that was created in exercise 18 and asking students if they have any ideas on how to improve the code. (The code repeats 3 times, so use a loop) • Present the code in exercise 19 (the code that was developed as part of exercise 18) • Ask students to find ways to simplify the code (If they need more clues ask them to come up with shorter ways to design the code) • Make sure students understand the notion of “repeating” code	

Part 1: 5 Minutes

Introduction Cont.

Discussion	5 mins.

- In the previous lessons we dealt with sequential programs and conditions.
- Oftentimes a program needs to repeat some blocks of code several times, so it is useful to use a loop.
- Ask students to identify a loop or repeating scenario in our game.
  - Guessing letters (players will keep guessing until they guessed all the letters or ran out of tries)
- To keep a computer working, we need repetition or to loop back over the same block of code again and again
- Ask the students what we need to define in order to run in a loop
  - Number of iterations
  - When to stop
- At this point, present only the concept and need for loops, you will go into the actual use of For loops later on in the lesson

Part 2: 35 Minutes

Playtime

Explanation – range()	**5 mins.**

The following explanation relate to exercises 19 - 20.

Introduce the range() function

- range() is a function that can get up to 3 parameters and it always returns a sequence of numbers
- range(num) returns a sequence starting from a default 0 with increments of 1 and up to num-1
  - For example, range(5) = 0,1,2,3,4
- range(start, stop) has a default increment of 1. It starts from the value of start and ends at the value of stop minus one.
  - range(3,8) = 3,4,5,6,7
- range(start, stop, increment) has no default value. The sequence starts with the start value and ends with the last value that should be in the sequence but is less than the stop value. The difference between all consecutive numbers in the sequence is the increment.
  - Example, range (3,8,2) = 3,5,7

Part 2: 35 Minutes

Playtime Cont.

Explanation – range() cont.	**5 mins.**

Ask students to write down the sequence:

- range(5) → 0, 1, 2, 3, 4

Ask students to write down the call that will return the sequence

- 5, 6, 7, 8→ range(5,9)

Ask the students to the write down the sequence:

- range(3,9,2) → 3, 5, 7

A range() function can also return an empty sequence - ask students to think of an example

- range(-5)

Part 2: 35 Minutes

Playtime Cont.

Explanation – For	**5 mins.**

- A **for loop** is used for iterating over a sequence
- A sequence could vary using numbers and strings
- Using **for loops,** we can execute a set of statements, once for each item in a list
- Each execution is called an iteration
- To loop through a set of code a specified number of times, we can use the range() function
- For example, ask students to write down the output of the following code:

```
for index in range(4):
    send_message(index)
```

- It is very useful to draw on the board a table to follow up on the code:

index	in range()	output
0	yes	0
1	yes	1
2	yes	2
3	yes	3
4	no	

Part 2: 35 Minutes

Playtime Cont.

Explanation – let()	**3 mins.**

How many guess will we let the player have?

- The same guesses as the number of letters in the word

How can we count the number of letters in the word?

- Using the **len()** function returns the number of characters (length) in a string

What will be the output of the following?

```
send_message(len("Python")):
```

The returned value of len() can be stored in a variable:

Ask - What will be the result of running the following instruction?

```
guessed_word = " ”
word=”game”
word_len = len(word)
for index in range(word_len):
guessed_word += "_ "
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – str()	**2 mins.**

Concatenating a string and an integer is not possible.

The function str() gets a number value as a parameter and returns its representation as a string.

```
my_var = 100
send_message("Fails " + my_var + "%!")
# Gives an error
send_message("Passes - " + str(my_var) + "%!")
# will display:
# Passes - 100%!
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 mins.**
Review log-in instructions here.	
Playtime	**25 mins.**
Have your class complete exercises 19 - 24. Encourage the students to go step by step. Pay attention to use the exact text as required in the exercise. Use your classroom's dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

Ask your class what the main topic of the lesson was:

- Loops, repeated execution
- For loops

What are loops good for?

Can students think of something from their daily life that is loop based?

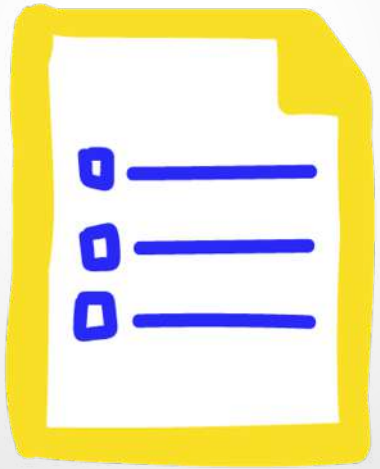
- Some examples: brushing your teeth (you brush all your teeth one by one), filling a water bottle until it is full (until loop) or eating a meal (you take bit after bite)

To build the loop, we used additional built-in Python functions. Which functions were used and for what purpose?

- Range – to define how many times the loop will occur
- Len – the length of the string that we go over its letters
- Str – Transferring an integer value into a string

Lesson 6 – Lists & Classes

This lesson is divided into 2 sections, the first part will introduce Lists and the second part will deal with building the game's Class.



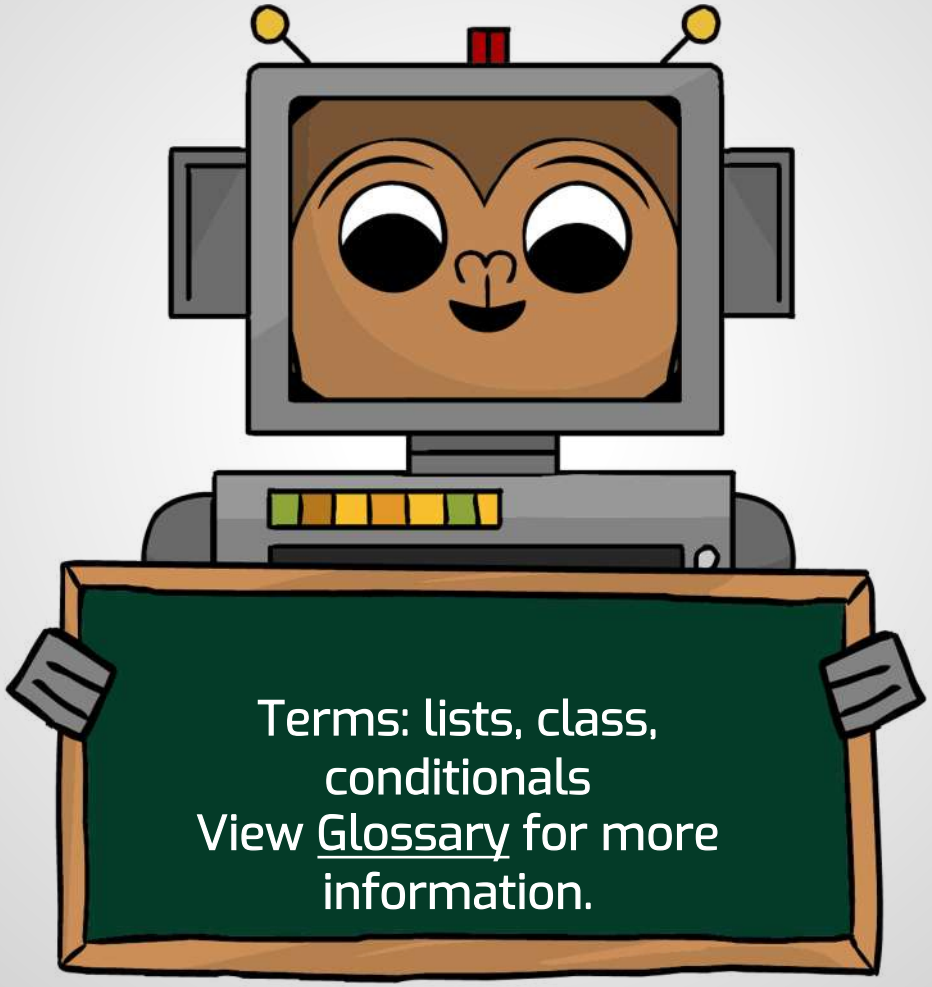
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Understand the concept of lists and what they are used for
- Manage lists
  - Holding multiple variables and values in one place
  - Access each value in the list
  - Add items to a list
- Complete exercises 25-30

Components



Part 1: 5 Minutes

Introduction

Activity	**4 mins.**

Challenge your students by asking them: What if we want to save all the guessed letters during the game?
Ask students to think of a way to save all the guessed letters. What do you think they will say to do if we have…

- 4 letters? You need 4 variables
- 10 letters? 10 variables
- 100 letters? Oh no, what can we do? The answer will be a list and add the values.

Can we use different variables to store them? While 4 or 10 can still be reasonable, 100 values require something else. This is where lists come in handy! **Please note**: lists are known as arrays in most programming languages

Part 2: 35 Minutes

Playtime

Explanation - lists	**4 mins.**

*Playtime can be done in parallel to the explanation
Refer to exercise 25
A list is a collection of variables which is ordered and changeable
To define a list:

```
my_list = [] # an empty list
my_list = ["a", "b"] #a list of two elements
```

To collect the guessed letters, we will use the Python append() function that adds an element to the end of the list

```
my_list.append("c")  #adds "c" to the list
send_message(my_list) #display: a,b,c
```

A useful real-life metaphor for lists are school lockers because they are ordered, the values can be changed and lockers can be added or removed when needed.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Lists (2)	**5 mins.**

Refer to exercise 26

The elements in the list are ordered/indexed sequentially starting from zero.

```
my_list = ["a", "b", "c"]
send_message(my_list[0]) #display: a
send_message(my_list[1]) #display: b
```

Ask students if the list has 4 items within it, what will be the index of the list item in the list? It will be 2. Lists hold values. Like in a building with a lot of apartments, there are numbers per apartment.

Part 2: 35 Minutes

Playtime Cont.

Explanation – int()	**2 mins.**

Refer to exercise 26

str() allows us to transform a numeric value into a string

Ask your class what happens if we have a string that represent a number and we wish to use it as a number?

- The int() method returns an integer object from any number or string - int() gets a string and if it represents a number, it returns its value as a number

```
my_string = "4"
my_number = int(my_string)
my_number += my_number
send_message( str(my_number) ) #display: 8
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 mins.**
Review log-in instructions here.	
Playtime	**8 mins.**
Ask the student to complete exercises 25 and 26 related to lists. Playtime can be done in parallel to the explanation Use the classroom dashboard to keep track of students' achievements.	

Part 2: 35 Minutes

Playtime Cont.

Explanation – Classes	**10 mins.**

- Our world cannot be represented with only simple variable types like strings or integers.
- Programming languages allow us to define more realistic types to represent the world we live in through classes.
- An example of a class in the real world is a form. There are many fields to fill out in the form (i.e. date, name, etc.). The blank form is a class definition. After you fill out the form, it becomes an object or instance.
- Instance − An individual representation of a certain class.

Part 2: 35 Minutes

Playtime Cont.

Explanation: Classes – Example 1	**10 mins.**

- This section corresponds with exercises 27-29
- The first line of the class is the name of the class
- The class Clock has 2 attributes - *hours* and *minutes*
- The first method __init__() is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class
- Initial values of the Clock-object attributes are defined in the init method - every new clock object created from the class clock will have the hours set to 12 and minutes set to 30

```
class Clock:
    def __init__(self):
        self.hours = 12
        self.minutes = 30
```

Refer to the current instance of the class

Part 2: 35 Minutes

Playtime Cont.

Explanation – Create Object	**10 mins.**

Exercises 27-29
The creation of the object is done outside of the class section
my_clock is the object name
Clock is the class
To access the attributes of an object we will use the object_name.attribute - (dot operator)

The dot operator is the connection between an object and one of its attributes or methods (functions).
For example: my_clock.hours
For example: my_clock.tick

```
my_clock = Clock()

send_message(str(my_clock.hours) + ":" +str(my_clock.minutes)) # display 12:30
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – Classes – example 2	**10 mins.**

- Exercises 27-29
- Initial values for an object's properties can be passed to the __init__(self) method.

```
class Clock:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – Create Object 2	**10 mins.**

- This section correlates with exercises 27-29

We created 2 clock objects - clock_1 and clock_2 - each has different values

```
clock_1 = Clock(14,45)

clock_2 = Clock(12,10)
```

Part 2: 35 Minutes

Playtime Cont.

Playtime	**10 mins.**

Have the class complete exercises 27 through 30

Go over ex. 27-29 with the 2 examples of clock and team and focus on ex #30 where the students has to build the game class with 2 attributes –

- Word to be guessed
- Letters guessed

Use your classroom's dashboard to keep track of students' achievements.

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

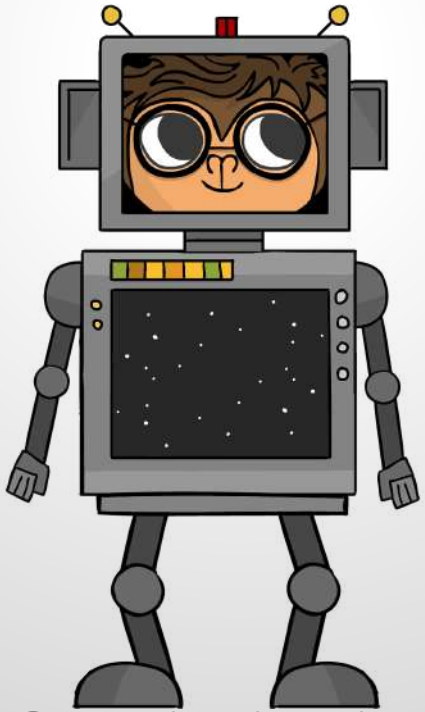
Summarize the lesson with recap of the 2 topics covered:

- Lists
- Classes & Objects

- Python is an object oriented programming language -
  Almost everything in Python is an object, with its properties and methods.
  A Class is a template for creating objects. An object is an item of the same type. A class is a definition of what features a clock, or any other object of that class, should have. A class for clock is hours, minutes and seconds and then we created different clock objects and each was initiated for a different time.
- It is important to mention that this is only the first time you touch the topics and the next lessons will include further exercises related to these important topics

Lesson 7 – Game - Class

This lesson will cover class methods.



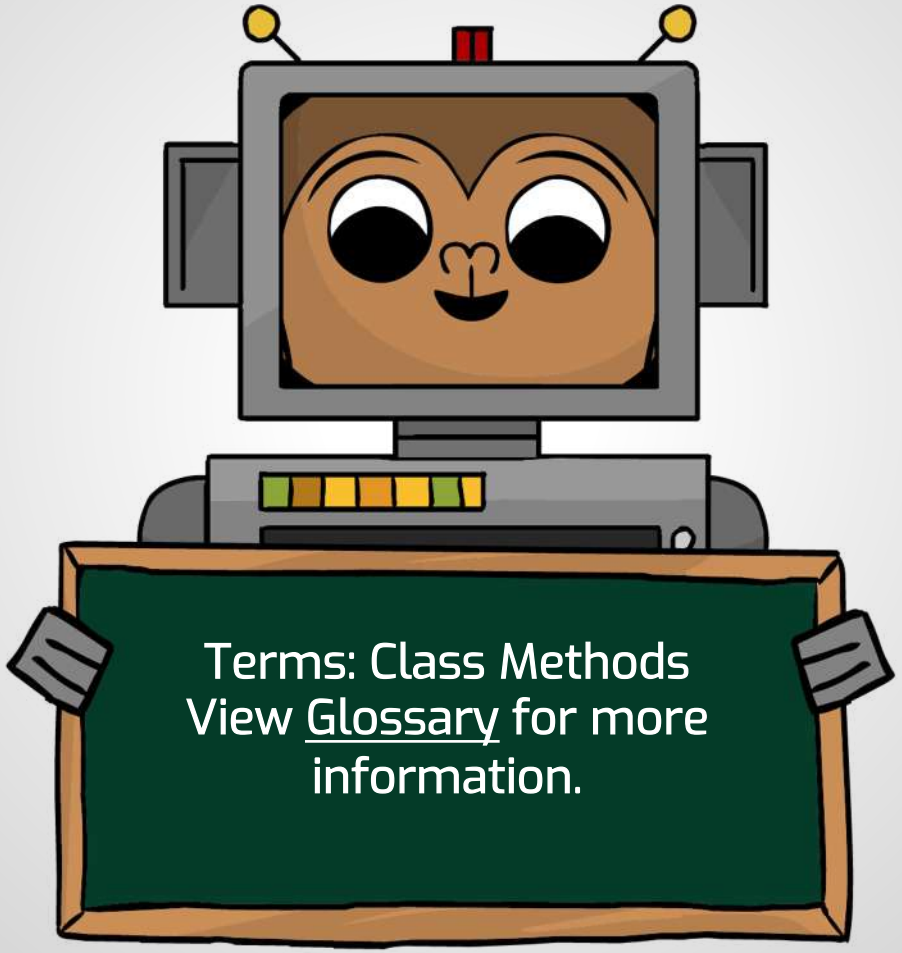
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Understand the concept of class methods and will learn how to define methods within a class
- Practice the use of for loops with strings
- Complete exercises 31-34

Components



Part 1: 5 Minutes

Introduction

Activity	**4 mins.**

Defining attributes for a class is only the first step, in order to manage the entire life cycle of the object created from the class, we need to have a set of methods to set, present, and manipulate the object . Classes can also contain methods. Methods in classes are functions that belongs to the object and can run when they are executed on a specific object.

Ask students to suggest methods that should be part of the class clock

- Set the time
- Show the time
- tick()

Tip: If students have a hard time understanding the concepts of classes and objects, let them define their own idea for a class with attributes and methods and create objects from the class.

Part 2: 35 Minutes

Playtime

Explanation – Class Method	**4 mins.**

Exercise 31

Define together with the students the method tick() within the class Clock

```
class Clock:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes
    def tick(self):
        self.minutes += 1
        if self.minutes > 59:
            self.minutes = 0
            self.hours += 1
        if self.hours > 23:
            self.hours = 0
```

Remember to add def in the beginning and a colon : at the end. Make sure the code you add is indented one tab.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Using a Method	**5 mins.**

Ask what is the value of clock and what will be presented?

```
clock = Clock(13, 23)
send_message(str(clock.hours) + ":" +str(clock.minutes))
# display: 13:23 (1:23 is the value)
clock.tick()
send_message(str(clock.hours) + ":" +str(clock.minutes))
# display: 13:24 (1:24 is the value)
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – Append()	**2 mins.**

append() is a list method that allows us to add an item to the end of the list

```
my_list=[”a”, “b”]
my_list.append(“c”)
send_message(my_list)
# display [“a”, “b”, “c”]
```

In order to build a list of the guessed letters, we will add each letter to the self.guessed_letters list by using the following instruction:

```
self.guessed_letters.append(guess)
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – For on String	**5 mins.**

Exercise 33

When we introduced the for loop we mentioned that it can also iterate over a sequence of characters - A string is a sequence of characters.

```
for char in "Python":
    send_message(char)
```

```
# display
P
y
t
h
o
n
```

Using for on string gives us the ability to go over the string and check each and every character, it will be very useful for the game

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 mins.**
Review log-in instructions here.	
Playtime	**23 mins.**
Ask the student to complete exercises 31 through 34. In exercise 34 pay attention to the nested loops - it is not recommended to focus on the topic at this point, however, for students who have previous background this could be an option for deepen their understanding on how nested loops works Use your classroom's dashboard to keep track of students' achievements.	

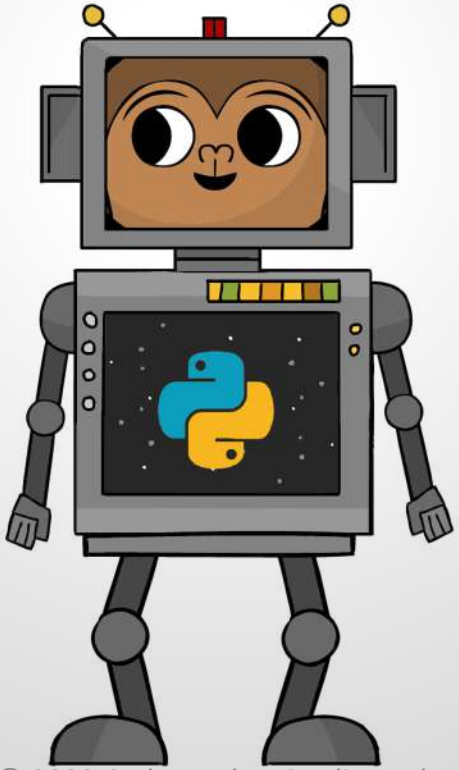
Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**
• Summarize the lesson with a discussion on the definition of class methods • This is a good point in time to draw attention to the fact that many of the functions they are using today on string and on lists are basically classes methods	

Lesson 8 – While & Boolean Operators

This lesson will cover how to use Boolean Operators.



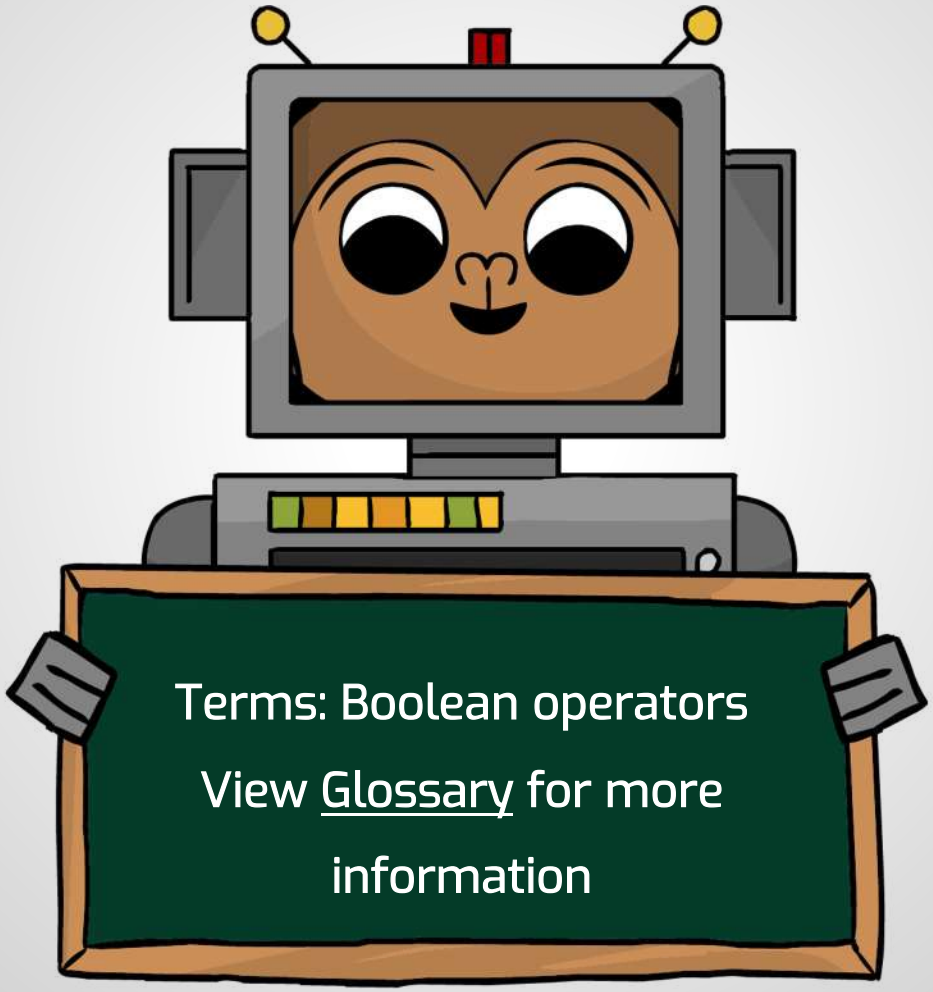
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Know how to user additional Boolean operators - ==,!=
- Understand the need and structure of the while loop
- Complete exercises 35-38

Components



Part 1: 5 Minutes

Introduction

Introduction – Code Review	**5 min.**

Today students will perform a peer-assessed code review.

- Ask students to think about the game and the way it was built like a product life-cycle in a high-tech company. In high tech, you design something, then you do a code review, a process to check the quality and design of the software. Today you will do this.

Ask students what are limitations of the game we designed it so far?

- Lead the students to think about the **number of guesses** allowed. The limitation is that you cannot guess an infinite amount of times.

At this point, the number of guesses is preset.

Is there a way we can change this situation?

- Can we do it with the **for loop**?

Part 2: 35 Minutes

Playtime

Discussion	**4 mins.**
The while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop requires relevant variables to be ready. Usually the value is changed during the loop statements.	

Part 2: 35 Minutes

Playtime Cont.

Explanation – Comparison Operators	**5 mins.**

- The following corresponds with exercises 35-38

Before we go into while loops, we need to learn how to create conditional statements

Comparison operators are used to compare two values, this lesson we will focus on 2 operators:

Operator	Name	Example
==	Equal - compares the values on both of its sides. The comparison's result is: True - if both values are exactly the same or False - otherwise	x==y
!=	Not Equal - used to check if values are not equal. The comparison result is: True - if values are not exactly the same or False - otherwise (the values are exactly the same)	x!=y

Part 2: 35 Minutes

Playtime Cont.

Explanation – Comparison Operators Cont.	**5 mins.**

Ask students which other comparison operators they can think of

- Mention greater than/smaller than

```
if "Python" != "nohtyP":
   send_message("not the same string")
else:
    send_message("strings are equal")
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – While	**5 mins.**

- The while loop is used when we can not determine the exact number of loop iterations in advance
- While loops repeat the sequence of actions many times until its condition value is False
- The condition is given before the loop body and is checked before each execution of the loop body
- In For loop, we always have the number of iterations, even if it is a parameter that we do not know its value in advance

Part 2: 35 Minutes

Playtime Cont.

Explanation – While Cont.	**5 mins.**

In the following while loop example we see the while structure

Line #1 - initializing the conditional variable (the conditional variable could be a number or string)

Line #2 - the conditional check

Line #3 - a block of statements

Line #4 - Usually the last line in the statements is updating the conditional variable for recheck in line #2

```
my_var = 3
while my_var != 0:
    send_message("my_var = " + str(my_var))
    my_var -= 1
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
Have students complete exercises 35 through 38. Encourage them to go step by step. Pay attention to use the exact text as required in the exercise Use your classroom dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

Discuss the difference between == and !=

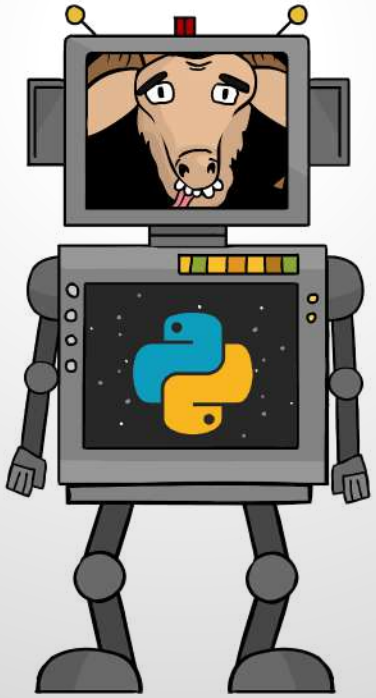
- 5==5 → returns True
- 5!=4 → returns True

Speak about the greater flexibility we have when using while loop

- No need to know the exact number of iterations before the loop starts
- Ability to run until a certain input is received

Lesson 9 – Advanced Operators

The main part of the lesson is aimed at introducing the students to Boolean/logical operator.

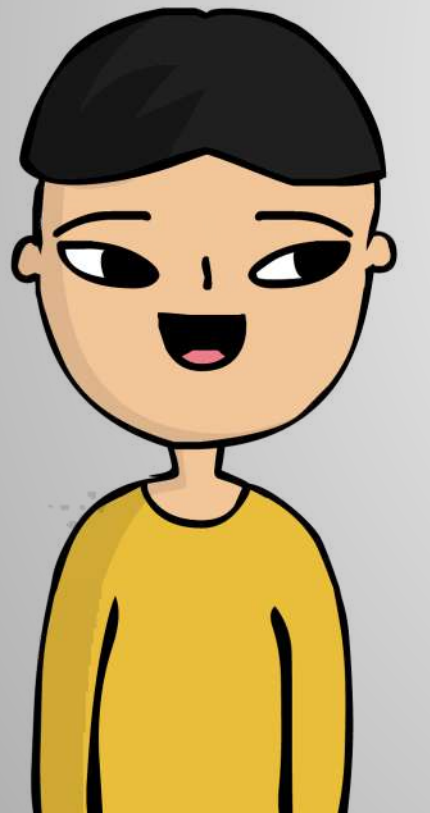


Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

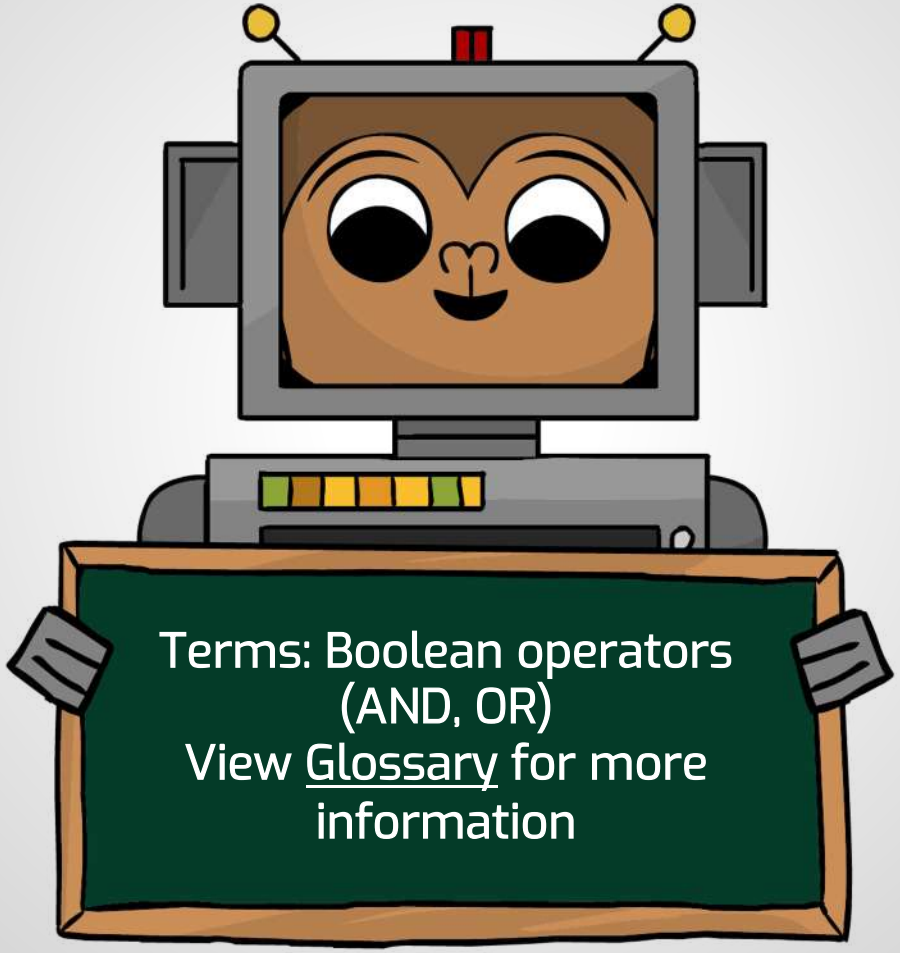
Objectives

Students will:

- Understand how to use more advanced conditional operators ‘not in’ and logical operators ‘and’ ‘or’
- Update the game to include advanced conditional checks that are based on logical operators
- Complete exercises 39-43



Components



Part 1: 5 Minutes

Introduction

Activity – Conditional Group	**5 mins.**
Divide the class into groups using AND and OR. • Boolean operator - There are five Boolean operators that can be used in programming to expand the conditional checks • At this point we will focus on AND and OR operators Ask students to split into 3 groups in accordance with the groups' definition written on the board Define the groups in a way that one group will be empty, one is Or and one is AND	

Designing the board		
Group A: You can join this group if you are older than 9 years old AND younger than 7 years old	Group B: You can join this group if you have long or curly hair	Group C: You can join this group if you are older than 8 AND younger than 11

Part 1: 5 Minutes

Introduction Cont.

Discussion	**5 mins.**

Ask the following questions:

- Can you join more than one group?
  - Yes
- Is there a group that no one could join? If so, why?
  - Yes, group A because it is impossible to fit both conditions
- What's unique about the empty group
  - both conditions return FALSE
- What is the difference between group B and C - the difference between AND and OR
  - In group B only one condition has to be met
  - Group C both conditions has to B met

Part 2: 35 Minutes

Playtime

Explanation – 'Not in'	**5 mins.**

- This explanation corresponds with exercise 39

Remind students the use of 'in' operator

- "P" in "Python" → True
- "a" in "Python" → False

How can we check if a string is '**not in**' another string?

The operator **not in** returns:

- **True** if the character is **not in** the string
- **False** otherwise

The operators **in** and **not in** have opposite values: when the value of one is True, the other is False and vice versa.

Definition : **Not in** (membership operator) – Has the value of True if a sequence with the specified value is not present in the object

Part 2: 35 Minutes

Playtime Cont.

Explanation – AND	**5 mins.**

How can we check if two conditions are met?

- By combining two conditions using the AND operator

The AND operator will return:

- **True** only if both of the values of the expressions (on either side of the operator) are True
- **False** in any other case

Let's define 2 variables and see how the AND operator works

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

- What will be the output of the following check?

```
send_message(x == 5 and y == 10)
# True and True is True
```

- What will be the output of the following check?

```
send_message(x == 5 and y == 5)
# True and False is False
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – AND cont.	**5 mins.**
The table summarizes the AND operator.	

x == 5	**y == 10**	**x == 5 and y == 10**
True	True	**True**
True	False	False
False	True	False
False	False	False

Part 2: 35 Minutes

Playtime Cont.

Explanation – OR	**5 mins.**

- This explanation corresponds with Exercise 42

How can we check that at least one of the conditions is met?

- Combining two conditions can be done by the OR operator

The operator OR will return:

True if at least one of the values of expressions (on either side of the operator) is True

False otherwise

Let's define 2 variables and see how the OR operator works

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

What will be the output of the following check?

```
send_message(x == 5 or y == 5)
# True or False is True
```

What will be the output of the following check?

```
send_message(x == 10 or y == 5)
# False or False is False
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – OR cont.	**5 mins.**
The table summarizes the OR operator, present it on the board together with the AND table so the students can see the difference.	

x == 5	**y == 10**	**x == 5 or y == 10**
True	True	**True**
True	False	**True**
False	True	**True**
False	False	False

Part 2: 35 Minutes

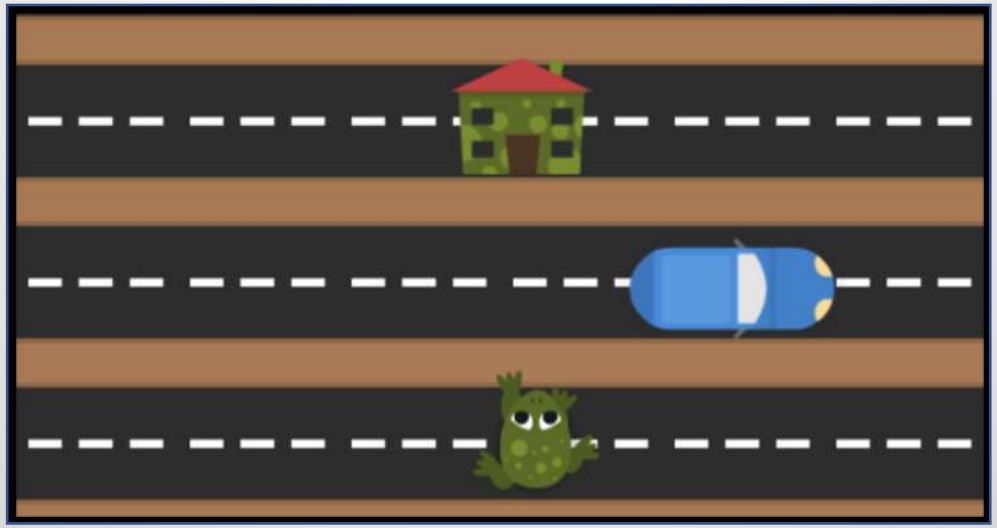
Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
Ask students to complete exercises 39 through 43. When discussing exercise 43, explain the conditional check that will be added to the game The game will be changed to allow the player to continue guessing only if the word is not guessed yet **and** there are more tries left The condition of the while loop needs to combine two separate conditions into one condition. Use your classroom dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**
Oftentimes in real life, we have to check multiple conditions before we perform an action. For example, before crossing the road, we need to make sure the traffic light is green and that the road is clear. Ask students for moments during the day where they use the conditions AND and OR	



Lesson 10 – Methods

The main part of the lesson is aimed at writing methods

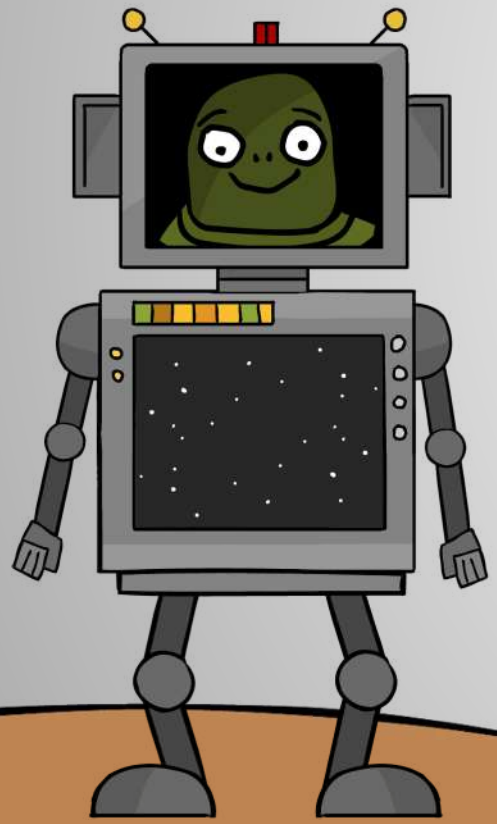
and using it in the code.

Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

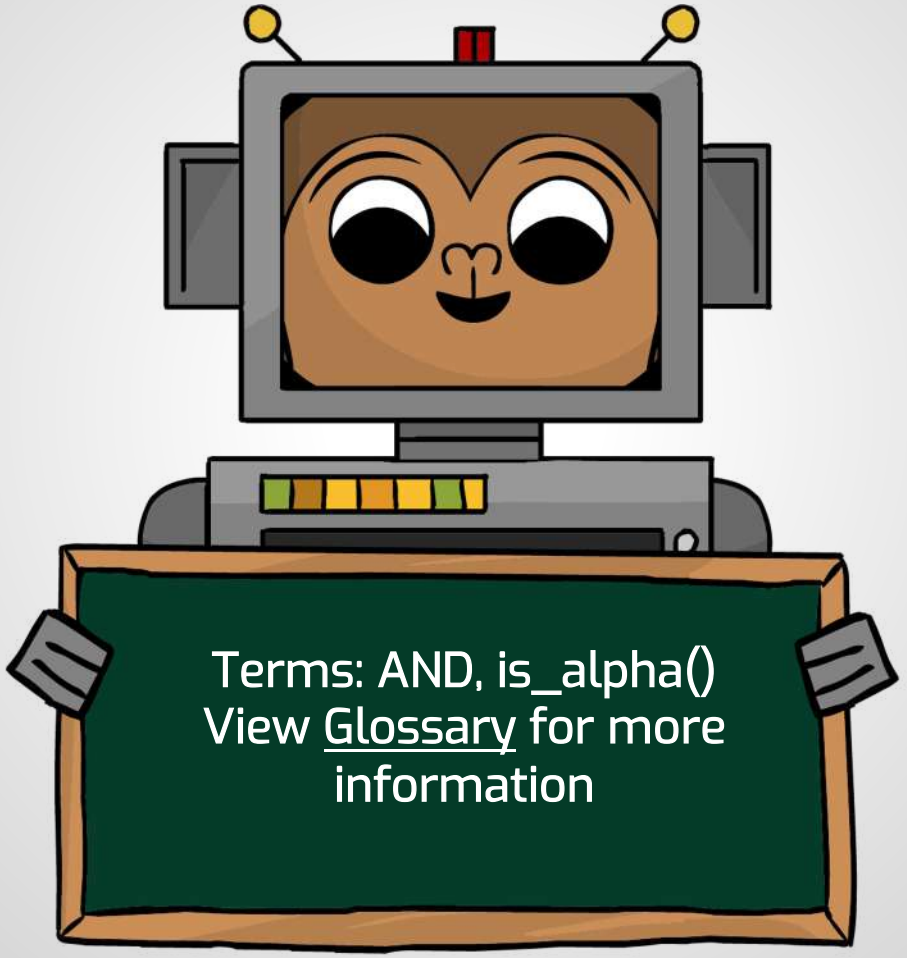
Objectives

Students will:

- Use the built-in method – is_alpha()
- Use AND operator for additional checks required for the game
- Design code using methods
- Complete exercises 44-48



Components



Part 1: 5 Minutes

Introduction

Explanations	**5 mins.**

- Methods is a named section of a program that performs a specific task. It is a type of procedure or routine.
- In Python there, is a distinction between a method and a function.
  - Both are called by its name, but a method it is associated to an object (dependent).
- A method is implicitly passed to the object in which it is invoked.
- Both methods and functions may or may not return any data (will be discussed in the next lesson)
- Why not write every program as one big ``chunk'' of statements?
- Programmers used to divide their programs into separate--but cooperating--methods
- Think back to the previous lessons where we used built-in Python methods - what if we didn't have len() or in?
- Those methods were predefined for you, meaning that you you could simply use those in your own program, without worrying about how the functions actually worked inside.

Part 1: 5 Minutes

Introduction Cont.

Discussion	**5 mins.**

The two most significant reasons that programmers use methods:

- **Reusability**: Once a function is defined, it can be used over and over and over again. You can invoke the same function many times in your program, which saves you time and work.
- **Abstraction**: In order to use a particular function you need to know the following things –
  - The name of the function
  - What the function does (Discuss with the class the difference between what the function does and how it does it)
  - What arguments you must give to the function
  - What kind of result the function returns

Part 2: 35 Minutes

Playtime

Explanation – 'is_alpha()'	**5 mins.**

The following explanation corresponds to exercises 44-45

- We would like to get the player to only guess letters
- Python give us built in tools to check if a string include letters only

```
my_var = "Python"
send_message(my_var.isalpha()) # True
my_var = "Python-101"
send_message(my_var.isalpha()) # False
```

Definition :is_alpha() returns True if a string includes only alphabetic characters and False otherwise.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Methods	**10 mins**

You added the exact same set of instructions to three different places in the code.

To avoid repeating the same code, create a method.

- A method can be viewed as a "new instruction”
- Its name should imply its purpose
- Instead of writing the set of instructions in the code, use the method's name
- Be sure to define a method in the class
    - Do not define a method inside a different method

Part 2: 35 Minutes

Playtime Cont.

Explanation – Methods cont.	**10 mins**

Which code in our program is repeated several times in different places?

```
send_message("Guess one letter at a time")
send_message("The word you are trying to find")
send_message("contains English letters only")
```

We can create a method that will display these instructions and call the method in each and every place the instructions are used

Define the method -

```
def display_instructions(self):
#write the code under the header
```

Once the method is defined, we can call it (use it) wherever we want in our class

Part 2: 35 Minutes

Playtime Cont.

Explanation – Unicode	**5 mins.**

Each character is represented by a unique integer and is kept in a table.

The two most used tables are:

- ASCII table (ASCII stand for: American Standard Code for Information Interchange)
- Unicode table

To print special characters, we will use the Unicode code of the characters -

You will use the Unicode for a heart → u"\u2764"

Part 2: 35 Minutes

Playtime Cont.

Explanation – Constants	**5 mins.**

Holding values in memory that can not change during the program

- You will use a constant that will hold the heart's value

Constants are defined in capital letters:

```
HEART_UNICODE = u"\u2764"
send_message(HEART_UNICODE)
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
Ask students to complete exercises 44 through 48. Make sure the students understand where is the right place to define the new method and where to call it within the game method Use your classroom's dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

Methods are used for 2 main reasons -

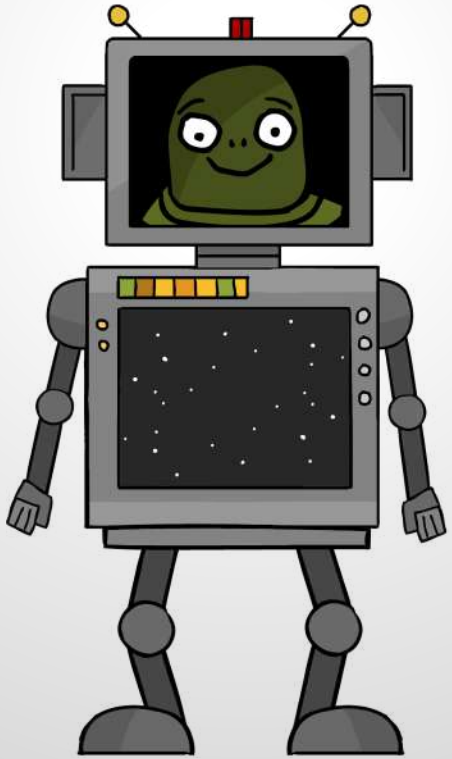
- Reusability
- Abstraction

It allows programmers to write more readable code

When writing methods, programmers plan and design their code in advance

Lesson 11 – Methods - Return Value

The main part of the lesson is aimed at writing a method that return values and using it within the code of the game.



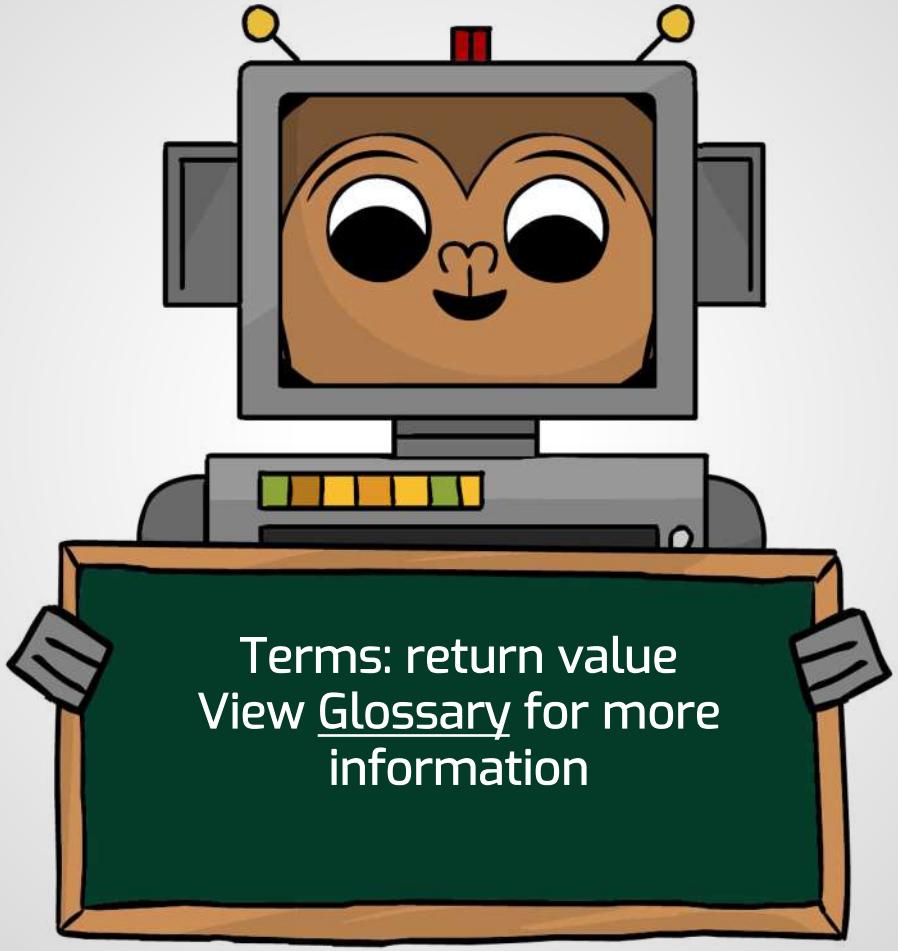
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Understand the difference between functions returning value and functions non-returning values
- How to user a return value and what type is could be
- Complete exercises 49-53

Components



Part 1: 5 Minutes

Introduction

Review	**5 mins.**

What is a returned value?

- The return statement causes your function to exit and hand back a value to its caller.
- The return statement is used when a function is ready to return a value to its caller.
- Function bodies can contain one or more return statement. They can be situated anywhere in the function body.
- Start the lesson with a recap of the previous lesson, ask the class:
- What are the two most important reasons that programmers use methods?

  Reusability - Once a function is defined, it can be used over and over and over again.

  Abstraction - use a particular method knowing how to call it and what will be performed, without knowing the way it is implemented

Part 1: 5 Minutes

Introduction Cont.

Discussion	**5 mins.**

- Ask the students if they see a way to improve the way methods works
- Can a method only print message?
- What happens if we want the method to calculate a value and use it somewhere else along the project?
- Students that have previous programming experience will probably mention the ability to return value from a method

Part 2: 35 Minutes

Playtime

Explanation – 'return'	**10 mins.**

Ex. 49 Look at the class Clock

```
class Clock:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes
    def get_time_formatted(self):
        time = str(self.hours) + ":" + str(self.minutes)
        return time
```

Use **chat** to run the code. What happens when we use the get_time_formatted() method?

```
clock = Clock(14,51)
my_time = clock.get_time_formatted()
send_message(my_time)
```

14:51

- Methods can calculate values and return the values to be used outside the method.

Part 2: 35 Minutes

Playtime Cont.

Explanation – 'return' cont.	**10 mins**

- The following corresponds with exercise 49

Return values can be assigned into a variable

```
my_time = clock.get_time_formatted()
```

Returned values can be of any type - string, int, bool, etc.

```
def is_today_sunday(day):
    if day == "Sunday":
        return True
    else:
        return False
```

Ask your students what the difference is between this method and the clock method - present the 2 different return statements (i.e. return true and return false) -

- Explain - only one of them will be executed
- Important - all code paths should return a value, if we get to the end of a method and we have not explicitly executed any return statement, Python automatically returns the value None.

Part 2: 35 Minutes

Playtime Cont.

Explanation – 'return' cont.	**10 mins**

A short and elegant way instead of using if else is:

```
def is_today_sunday(day):
    return day == "Sunday"
```

The returned value can be the result of a conditional statement or an arithmetic statement

There is no need to use a variable to hold the returned value

You can return the value of this condition (either true or false so you do not need an if statement)

Part 2: 35 Minutes

Playtime Cont.

Explanation – 'Calling a method'	**5 mins.**

The following explanation correlates with Exercise 52.
Using a method that returns a value can be done in two ways.
This is the method:

```
def sum_of_one_plus_ten():
        return 1 + 10
```

1. One way is assigning the returned value of the method into a variable:

```
sum = sum_of_one_plus_ten()
send_message (sum)
```

2. Another way is using the method as an argument of another function:

```
send_message( sum_of_one_plus_ten() ) #display 11
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	20 mins.
• Ask your students to complete exercises 49 through 53. • This lesson focuses mainly on building new methods out of existing code and calling the methods. • Make sure students understand where to write the new methods and how to call them. • Important - remind students when and how to use self - once as the method attribute and once when calling the method • Use your classroom's dashboard to keep track of students' achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

Methods can return a value of any type

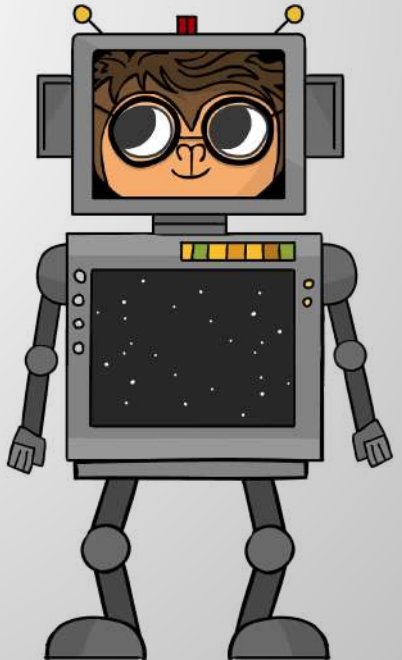
- Int, string, bool

Returned values can be an expression, conditional expression or an arithmetic calculation

- Calling a method is done in multiple ways
- Assigned into a variable
- A function can be used as an argument of another function
- Methods (within a class) will get at least one attribute (self) and will be called using self.method_name()

Lesson 12 – Methods, Import

The first part of the lesson is aimed at improving and enhancing some of the methods that already exist in the game by using the return mechanism. The second part of the lesson will focus on enhancing capabilities by importing python modules.



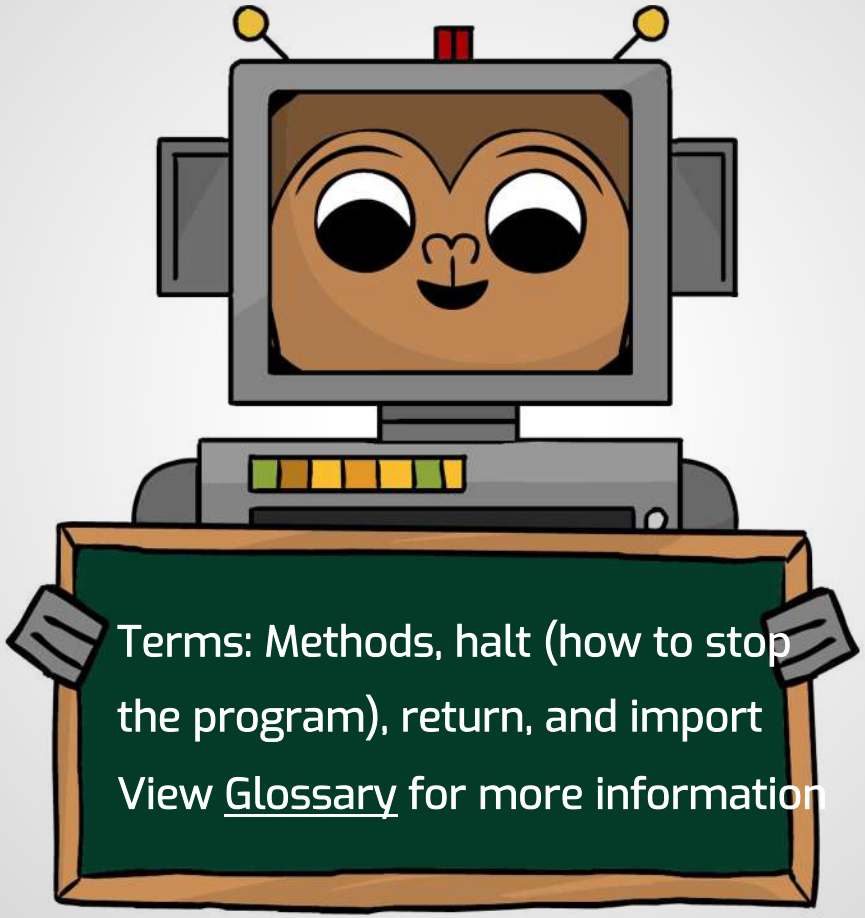
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Learn how to use existing Python modules to enrich their code
- Improve their ability to design and develop methods
- Complete exercises 54-58

Components



Part 1: 5 Minutes

Introduction

Activity – Code Review	**5 mins.**

- It is time to take a look at what we have achieved so far
- Start the lesson with presenting the project:
  - We have the class Game and several methods within it

```
class Game:
    def __init__(self, word):
    def play(self):
    def display_instructions(self):
    def display_hearts(self):
    def is_valid_guess(self, text):
    def get_dashed_word(self):
    def handle_valid_guess(self, guess):
```

Part 1: 5 Minutes

Introduction Cont.

Activity – Code Review Cont.	**5 mins.**

- Ask students to use their own words to define the purpose of each method
- See below a list of methods with their description

Method	Purpose
def __init__(self, word):	Game class constructor
def play(self):	Main method - running the game
def display_instructions(self):	Game instructions, presented to the player at the beginning of the game
def display_hearts(self):	Display a row of hearts - according to the number of guesses left
def is_valid_guess(self, text):	Checking the player's guess
def get_dashed_word(self):	Returning a string of dashes
def handle_valid_guess(self, guess):	Method to manage the process of guessing a valid letter

Part 2: 35 Minutes

Playtime

Explanation – 'halt'	**10 mins.**

- The following explanation corresponds with exercise 54
- Sometimes we would like to stop the execution of a method in the middle

Let's look at handle_valid_guess method

- Guessing the same wrong guess more than once reduces tries each time it is guessed. Seems rather unfair!
- We can fix it by stopping the execution when the letter guessed was already guessed before.

```
def handle_valid_guess(self, guess):
self.guessed_letters.append(guess)
        if guess not in self.word:
            self.tries_left -= 1
            self.display_hearts()
```

Part 2: 35 Minutes

Playtime Cont.

Define – 'Return'	**10 mins**
The following corresponds to exercise 54: A return instruction can be used to stop the execution of a method. When the execution of a code encounters a return instruction, it returns to the place it was called from and returns a value (either a specified one or None).	

Part 2: 35 Minutes

Playtime Cont.

Explanation – Modules Import	**10 mins**

The following corresponds to exercises 56 and 57.

Programmers are known for loving to reuse existing solutions

- Python gives us libraries of standardized solutions for many problems that occur in everyday programming.

Such a library is called a module:

- A module is a file containing Python definitions and statements.
- A module can be used in another program or module by importing it
- The file name is the module name with the suffix .py appended.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Build your own module	**5 mins.**

What happens when the code gets longer?

- As your program gets longer, you may want to split it into several files for easier maintenance.
- You may also want to use a handy function that you've written in several programs without copying its definition into each program.
- To support this, Python has a way to put definitions in a file and use them in a script.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Import String	**5 mins.**

The following correlates with exercise 56

- **String module** - The string module contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings
  - You need to import the string module in order to use any of its code
  - Add the following instruction at the beginning of the code:
    - import string

Let's take a look at the the constant named **ascii_lowercase,** which holds all the lowercase alphabetical letters

Other Python libraries, for example, include the **datetime** module which has class definitions for manipulating dates and times in both simple and complex ways.

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	20 mins.
• Ask students to complete exercises 54 through 58. • Exercises 54 and 55 are focused on improving existing methods using Return • Exercises 56 and 57 are focused on using an external module – string • In exercise 58 students will practice modularity by grouping code into a method • Use your classroom's dashboard to keep track of students' achievements.	

Part 3: 5 Minutes

Debriefing

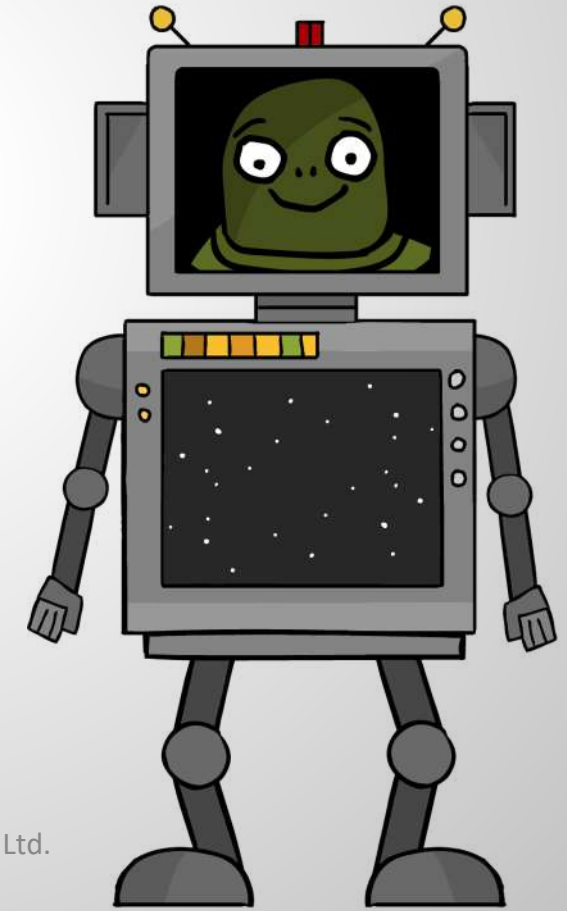
Debriefing	**5 mins.**

In this lesson we learned how to become better programmers, but how exactly?

- Using return to stop methods when their mission is accomplished
- Using external libraries or splitting our code into files for easier management of projects

Lesson 13 – Modulo & Booleans

In this lesson, students will further practice Boolean Operators.



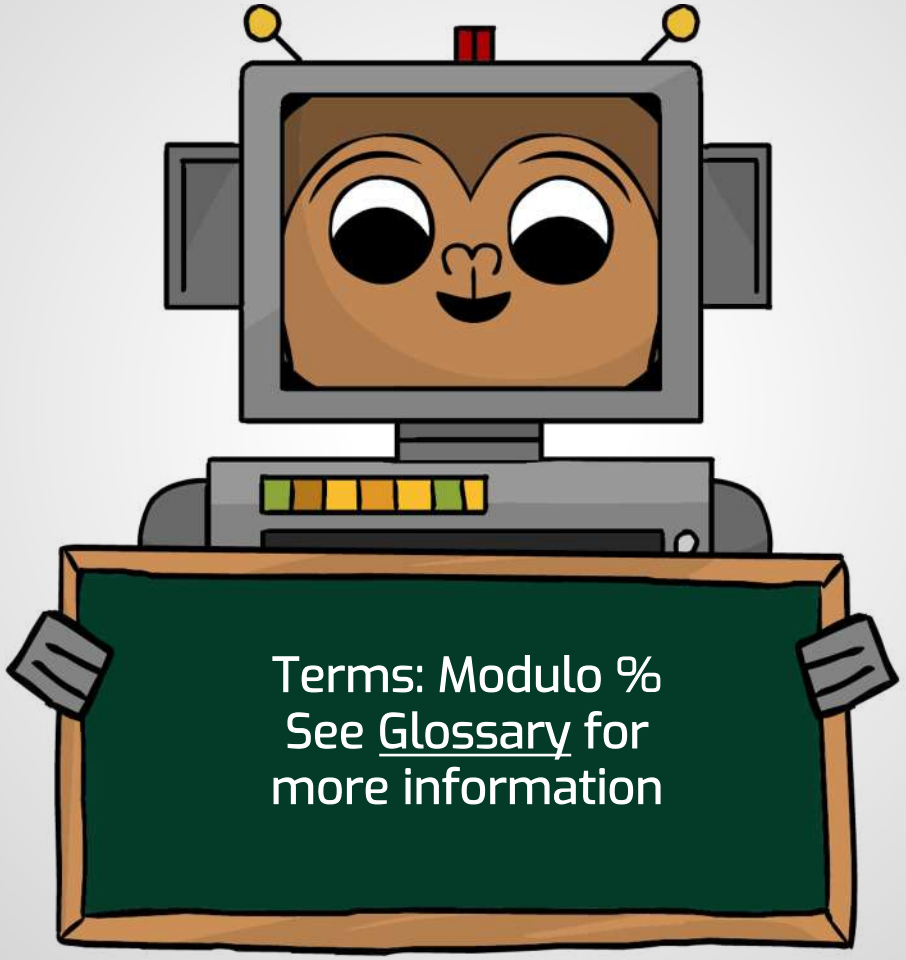
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Practice advanced Boolean values and statements
- Practice modulo operator
- Complete exercises 59-62

Components



Part 1: 10 Minutes

Introduction

Activity – Modulo and Remainders	**5 mins.**

Bring a box of candies, cards or any other object that you could distribute to the students only as a whole.
You should have enough items to hand out to the entire class plus one or two that will be left over.
Write on the board an exercise - (number of items) divided by (number of students)

E.g. 22/18

The answers will probably be a rational number

Now, ask one of the students to start dispensing the items you brought to the rest of the class to show what the remainder is

Part 1: 10 Minutes

Introduction Cont.

Discussion – Modulo Operation	**5 mins.**

- The modulo operation finds the remainder after division of one number by another.
- Boolean values are the two constant objects: False and True.
- A numeric value can be evaluated as a True/False statement.
- Zero is False any other number is True

Part 2: 35 Minutes

Playtime

Explanation – 'mod %'	**5 mins.**

How can we combine the mod operator with an if statement?

- The result of the statement yields an integer number

Use it as a Boolean variable

- 0 – false
- Any other number - true

Part 2: 35 Minutes

Playtime Cont.

Define – 'Modulo'	**10 mins**

In computing, the modulo operation finds the remainder after division of one number by another

- For example, the expression "5 mod 2" would evaluate to 1 because 5 divided by 2 leaves a quotient of 2 and a remainder of 1
- Ask - what is the possible result of X mod 2?
- Lead the students to define the general answer - even mod 2 is 0, odd mod 2 is 1

Part 2: 35 Minutes

Playtime Cont.

Explanation – Boolean Variables	**10 mins.**

- The following corresponds with exercise 61
- A variable can hold a Boolean value.
- True or False and be used used for decision making without any operators
- In the following example, daylight savings is a variable that is initialized to True

```
class Clock:
    def __init__(self, hours, minutes):
        self.hours = hours
        self.minutes = minutes
        self.daylight_saving = True
    def get_hour(self):
        if self.daylight_saving:
            time = self.hours + 1
        else:
            time = self.hours
        return time
clock = Clock(19,47)
send_message( clock.get_hour() ) # 20
```

Part 2: 35 Minutes

Playtime Cont.

Define – Boolean Variables	**5 mins.**

Boolean variables are used for decision making -
Any object can be tested for truth value, for use in an if or while condition or as operand of the Boolean operations below. \
The following values are considered false:

- None
- False
- zero of any numeric type, for example, 0, 0L, 0.0, 0j.
- any empty sequence, for example, '', (), [].
- any empty mapping, for example, {}.

All other values are considered true — so objects of many types are always true.

Part 2: 35 Minutes

Playtime Cont.

Explanation – Not	**5 mins.**

- The following corresponds with exercise 62
- The Boolean NOT operator inverts the value of a Boolean expression.
    - Note - for students who have previous experience in in java or C#, Python not is equivalent to !

value	**Not value**
true	false
false	true

- Not can be used in if statements or in other instructions. For example:

```
def is_not_five(number):
    return not number == 5
send_message(is_not_five(3)) # True
send_message(is_not_five(5)) # False
send_message(is_not_five(0)) # True
```

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
• Ask the student to complete exercises #59 to #62. • Explain again the mod operator when using integer numbers • Use your classroom's dashboard to keep track of students' achievements.	

Part 3: 5 Minutes

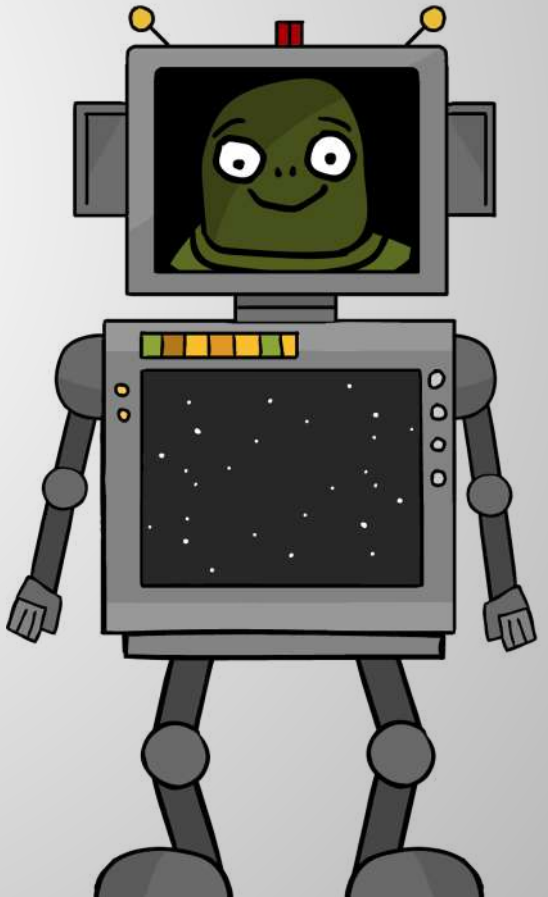
Debriefing

Debriefing	**5 mins.**

- In this lesson we learned 3 important programming tools and tricks
  - Modulo
  - Numeric values for true and false
  - Not operator
- To recap the class ask the students to give one example for each topic and how to use it in code

Lesson 14 – elif

Students will improve their code and learn about the elif structure and used it for improving the game.



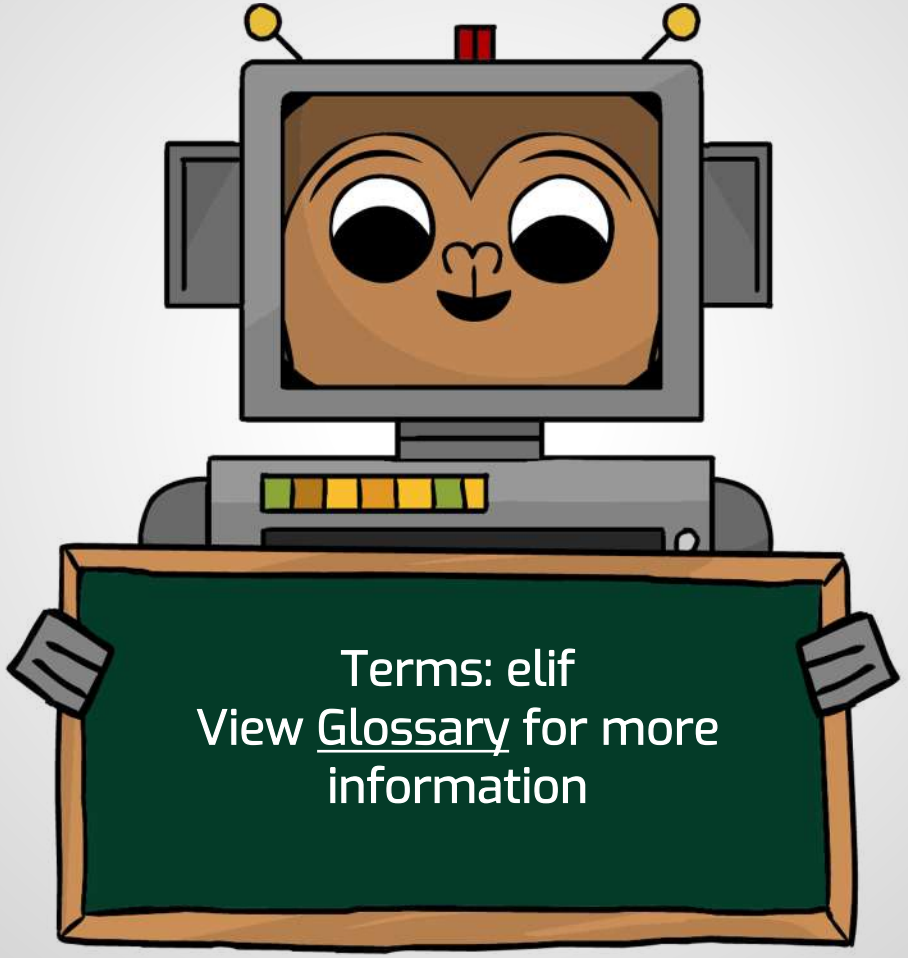
Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives

Students will:

- Write advanced conditional statements using elif
- Improve code readability and modularity using quit and splitting code into additional methods
- Complete exercises 63-66

Components



Part 1: 5 Minutes

Introduction

Activity	5 mins.

- Ask your students to write a code that checks the user's birth season - the user's birth season will be received as an input (as a char/number 1-fall, 2-winter, 3-spring or 4-summer). A relevant message will be displayed
- Give students 2-3 minutes to work in pairs and ask for suggestions
- Since we are assuming that they do not have previous Python background, they will use if and else with nested if statements.
- At this point, go directly to introducing the elif structure

else if structure	elif Structure
If it is Sunday, Go to gym Else if it is Tuesday Go teach Else if it is Friday Prepare the Dinner Else Meet with friends	If it is Sunday, Go to gym elif it is Tuesday Go teach elif it is Friday Prepare the Dinner Else Meet with friends

Part 2: 35 Minutes

Playtime

Present and Explain	**5 mins.**
An elif statement can replace multiple if checks. It gives a more descriptive way to compare a value with multiple variants.	

Part 2: 35 Minutes

Playtime Cont.

Define – 'elif'	**5 mins**

- The following corresponds with exercises 64 and 65
- The elif statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
- Similar to else, an elif statement is optional. However, unlike else, for which there can be at most one statement, there can be an arbitrary number of elif statements following an if.

else if Structure	**elif Structure**
If it is Sunday, Go to gym Else if it is Tuesday Go teacha Else if it is Friday Prepare the Dinner Else Meet with friends	If it is Sunday, Go to gym elif it is Tuesday Go teach elif it is Friday Prepare the Dinner Else Meet with friends

Part 2: 35 Minutes

Playtime Cont.

Define – 'elif' cont.	**5 mins**

```
if expression1:
  statement(s)
elif expression2:
  statement(s)
elif expression3:
  statement(s)
else:
  statement(s)
```

Part 2: 35 Minutes

Playtime Cont.

Explanation – Code Improvement	**5 mins.**

- The following correlates with exercise 63
- The player should have an option to quit the game at any point in time
- We have the flexibility to add new properties to our Game class
- This is a good opportunity to discuss the use of properties and the difference between properties and class methods
- Adding a new property to handle ‘quit’ input is a mutable way to handle exceptions

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
• Ask students to complete exercises 63 through 66 • These are the last stages of the game so it is a good time to work with students who are still struggling with coding on how to review code • Use your classroom's dashboard to keep track of student achievements	

Part 3: 5 Minutes

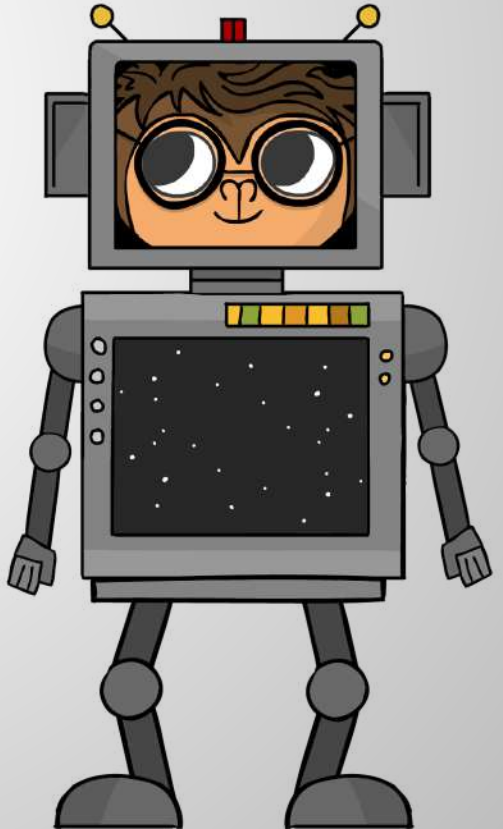
Debriefing

Debriefing	**5 mins.**

- In this lesson we learned the elif structure and used it for improving the game
- At this point of the course, most of the code already exists and it is time for improvements

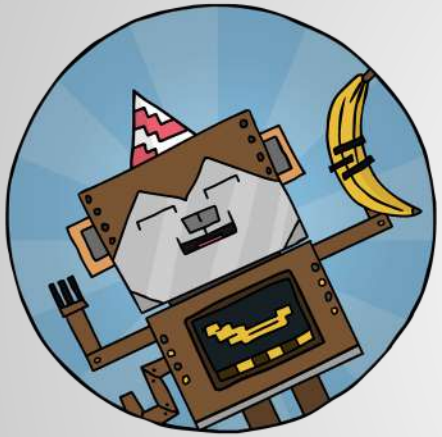
Lesson 15 – Improvements - Lists and more

The following lesson will cover what makes a good code design and discuss lists as a fundamental structure in programming.



Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives



Students will:

- Improve the game flow
- Learn about a new data structure called lists
- Complete exercises 67-70

Components

Part 1: 5 Minutes

Introduction

Activity - Lists	**5 mins.**

Today's activity will help students understand what purpose lists serve in programming.
Start off by asking the class how they or their parents prepare for grocery shopping. They should think of a list
Ask the following questions:

- What is a list?
  - A collection of elements/words/objects that are sorted by order of appearance, importance, alphabetically, etc.
- What can you do with you list?
  - Add items, delete items, check what's still missing, mark out of stock items, etc.
- What does grocery shopping have to do with coding?
  - Lists are one of the most used datatypes in Python and are very flexible. Let's see how we can use it on our game to improve the flow.

Part 2: 35 Minutes

Playtime

Discussion	**5 mins.**

Challenge students with the following question:

- Assuming the players would like to play the game again, how can we select the word to guess in each and every round?

Lead the discussion

- We would like to have various, different words ready for game runs
- We cannot add words during the game
- Assuming we would like to have 100+ words available for us, how can we manage them?

Summarize with the need for more advanced/complex data structures

Part 2: 35 Minutes

Playtime Cont.

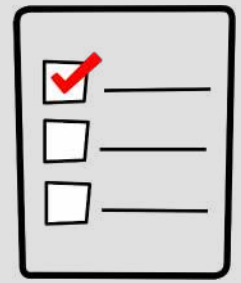
Define – 'lists'	**5 mins.**

List is one of the simplest, most important and flexible data structures in Python.
They are used to store and manage multiple values in a single list. Each item in the list has an assigned index value.
A list is mutable, meaning you can change its content.
An important thing about lists is that items in a list do not need to be of the same kind.
Creating a list:

- Lists are enclosed in square brackets [ ] and each item is separated by a comma.

For example:

```
words = ["snowman", "bookkeeper", "spelling", "princess", "wonderful", "playing", "marvelous"]
```



Part 2: 35 Minutes

Playtime Cont.

Explanation – lists	**5 mins.**

The following corresponds with exercise number 69.
Use Coding Chatbots to present the use of a list and use chat to run your code

- Creation of a list
  - words = ["snowman", "bookkeeper", "spelling", "princess", "wonderful", "playing", "marvelous"]
- Accessing items from the list
  - Words[0] → snowman
- Updating the list
  - words[0]=”game” → ["game", "bookkeeper", "spelling", "princess", "wonderful", "playing", "marvelous"]

Part 2: 35 Minutes

Playtime Cont.

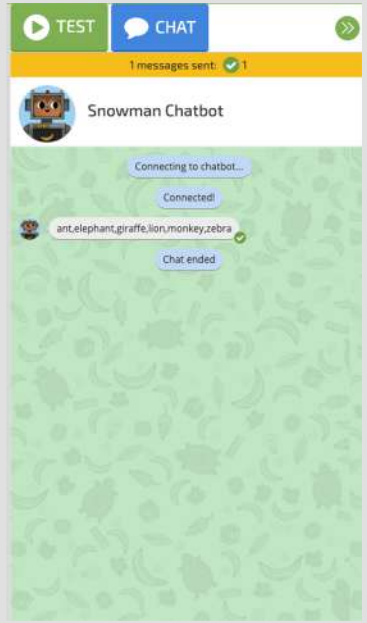
Explain – list operations	**5 mins.**

Exercise number 70

- The real benefits of a list are driven from the Python built-in list operators like append() and remove()
- This lesson will focus on append() and remove()
- To present the capabilities, use the chat button
    - list.append(obj)
      Appends obj to list
    - list.remove(obj)
      Removes obj from list
- Present the following code:

```
my_list = ["coding", "is", "not", "fun"]
my_list.remove("not")
send_message(my_list)
```

- Ask your students what the output will be.



Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	20 mins.
• Have your students complete exercises 67 through 70. • Exercises 67 and 68 improve the game • The improvement is done by using the while mechanism that was introduced in earlier lessons • Exercises 69-70 improve the game further using a list of words • Use your classroom's dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

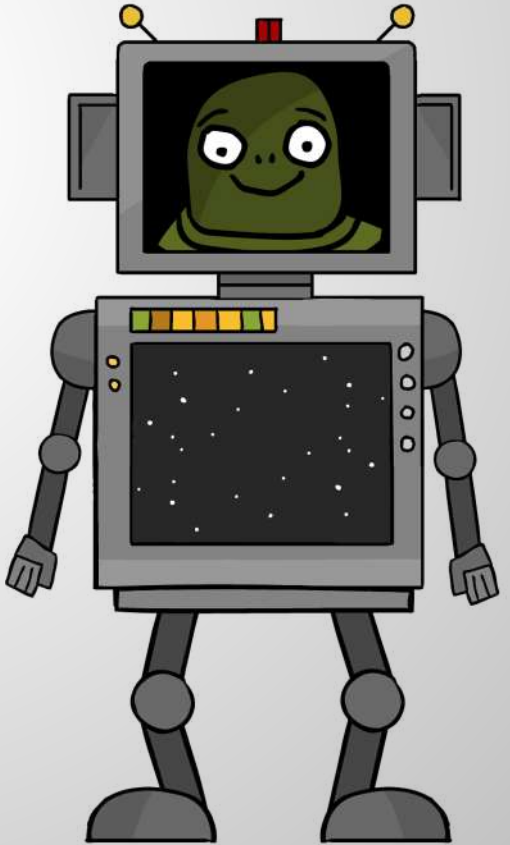
We are almost done. In the last couple of lessons we improved the game flow. Use this opportunity to discuss what a good code design is

- Speak about modularity, readability, efficiency, and ease of changes.
- The way we designed the game allows us to change it with new requirements

Recap the lesson by discussing lists and how they are a fundamental structure in programming

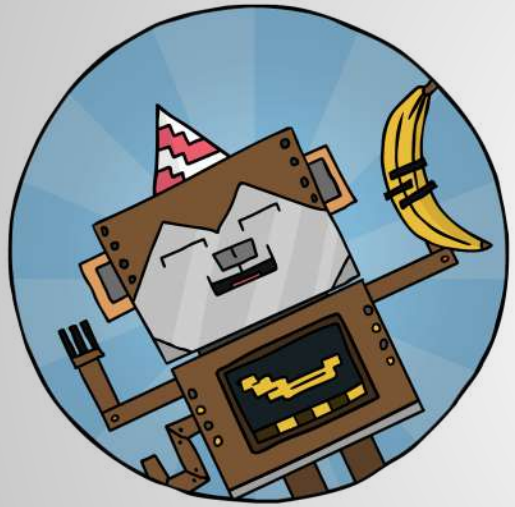
Lesson 16 – More elif

This lesson will focus on finishing touches in the game.



Please note that the minimum class time required for this lesson is 45 minutes. If you are rushed for time, please skip to Part 2.

Objectives



Students will:

- Have completed programming a complete, running guessing game
- Complete exercises 71-74

Components

Part 1: 5 Minutes

Introduction

Activity	**5 mins.**
Ask students to come up with idea for improving the game. • One way to improve the lesson is by randomly defining words to guess	

Part 2: 35 Minutes

Playtime

Discussion	5 mins.

- With the help of the class, list different ideas for improving the game on the board
- Explain the life-cycle of code/projects - projects that are well-written are easier to improve and adapt
- Ask students what happens when a players wants to play again and again and again?
    - After several rounds, the variety of words will come to an end
    - How can we improve this?

Part 2: 35 Minutes

Playtime Cont.

Define – Random	**5 mins**
Random is a built-in python module that needs to be imported before utilization. Using the random class, you can generate random numbers within a range. random. randint(low, high) generates a random number between the low and high numbers	

Part 2: 35 Minutes

Playtime Cont.

Explanation – Random	**5 mins.**

The following corresponds with exercise number 71
• Ask the class how you can imitate a rolling die
• You need to generate random numbers between 1 to 6
• How does it work?
`import random`
`dice = random.randint(1, 6)`
`send_message(dice)`	

Part 2: 35 Minutes

Playtime Cont.

Define – Choice	**5 mins.**
choice() – a method within the Random class that selects a random element from a list – to use it you use it with the random class according to the following structure random.choice(<list_name>). Each item in the list has an equal probability of being chosen. Return a random element from the non-empty sequence (or seq for short). If seq is empty, it raises IndexError.	

Part 2: 35 Minutes

Playtime Cont.

Explain – list operations cont.	**5 mins.**
• The following corresponds with exercise number 74 • Use choice() to randomly select a word from a list word=random.choice(words_list) • word will be assigned a random element from the list words_list	

Part 2: 35 Minutes

Playtime Cont.

Log-in	**1 min**
Review log-in instructions here.	
Playtime	**20 mins.**
• Have students to complete exercises 71 through 74. • Start with exercises 71 and 73 to learn the random syntax and how to use it • Exercises 72 & 74 are used for implementing the new instructions in the game • Use your classroom's dashboard to keep track of student achievements.	

Part 3: 5 Minutes

Debriefing

Debriefing	**5 mins.**

- The game is done and the course was successfully completed
- Use this opportunity to praise the students for their progress and determination to learn and work
- Ask students to change the words in the list and invite them to switch places so each one can play their friend’s games

Glossary

Term	Description	Example
-	Arithmetic Operators are used to perform calculations like addition, subtraction, and multiplication. - is the Subtraction Operator. In order to subtract, write A - B, where the minus sign is the operator that performs the subtraction and A & B represent the operands that are subtracted from one another. This operator can be binary, where it has two operands, or unary, where it has only one operand.	5 - 3
+	+ is the Addition Operator. In order to add, write A + B, where the plus sign is the operator that performs the addition while A and B are the operands that are being added together. Other than in addition, the “+” sign is used to concatenate, or combine, strings. In order to merge two strings into a single object, you may use the “+” operator.	2 + 3 language = "Python" game = "snowman" send_message(language + " " +game) #Python snowman is displayed. " " is a space concatenated.
+=	Short addition/Concatenation - used for shorter instructions to add values or concatenates string.	name = "Pyth" letter_1 = "o" letter_2 = "n" name = name + letter_1 # "Pytho" name += letter_2 # "Python"
-+'	Subtracts from the variable and later assigns the same value. This operator cannot be used with strings.	x-=y is equivalent to x=x-y

==	This is the ‘is equal to’ command. It compares the values, or operands, on both sides. The comparison's result is true if both values are exactly the same or false if otherwise.	language = "python"
		if language == "Python":
		send_message("Cool!")
		else:
		send_message("Oops..")

!=	This is the ‘not equal to’ command. It is used to check if the values are unequal. The comparison's result is true if the values are not exactly the same or false if the values are exactly the same.	if "Python" != "nohtyP":
		send_message("not the same string")
		else:
		send_message("strings are equal")

Glossary

Term	Description	Example
%	This is the Remainder or Modulo Operator. In computing, the modulo operation finds the remainder after dividing one number by another.	5 % 6 yields 3, which is the remainder of dividing 15 by 6 odd number % 2 yields 1 even number % 2, yields 0

init	All classes have a function called __init__(), which is always executed when the class is initiated. Use the __init__() function to assign values to object properties.	class Clock:
		def __init__(self):
		self.hours = 12
		self.minutes = 30

and	'and' is an operator that will be true if both of the operands (x and y) are true.	x = 5 (x > 0 and x < 10) # True (x > 0 and x > 10) # False
append()	This command adds an element to the end of the list. To use append, write your list, add a period and then use append()	my_list.append("c") # adds "c" to the list
ascii_lowercase	This constant holds all of the lowercase alphabet letters. It is defined in the string module.	send_message(string.ascii_lowercase) # "abcdefghijklmnopqrstuvwxyz"

Glossary

Term	Description	Example
Boolean	A conditional statement that always has a value of either ”Tue” or “False”. A variable can hold a Boolean value. Boolean variables are used for decision making.	5==4 ? False 5!=4 ? True
char	A variable holding a single character that can be used as a for loop range variable.	for char in "Python": send_message(char)
Chat	Chat Mode is used to test your code and acts similarly to a Python compiler. Chat causes the messages to be sent. When a read_message is reached in the code, the execution is halted and the chatbot waits for the user to respond. Once the user responds, the chatbot continues running.	
choices()	Chooses a random item from a list.	my_list = ["coding", "Python", "is", "fun"] item = random.choice(my_list)

Class	Classes are essentially a template to create your objects. In a class, we will define the object's properties and methods.	class Clock:
		def __init__(self):
		self.hours = 12
		self.minutes = 30

Conditionals	Statements that only run under certain conditions. A conditional statement returns only True or False.	if-else

Glossary

Term	Description	Example

elif	elif is a combination of else and an if statement that comes right after it. elif is used to distinguish different actions to different conditions and is used to avoid excessive indentation.	if expression_1:
		# code_1 block
		elif expression_2:
		# code_2 block
		else:
		# code_4 block

else	An if statement can be followed by an else statement.
	If the condition is true, the code inside of if is executed, if the condition is false, the code inside else is executed.
	Make sure you add a semicolon (:) after else and indent the block instructions .
		if "x" is in "Python":
		send_message("x in Python? This can not be!")
		else:
		send_message("x is not in Python")

For Loop	The for loop in Python is used to iterate, or repeat, over a sequence. A loop is the act of doing something over and over again.	for index in range(4): send_message(index) #the following block will print the numbers 0 to 3
Function	Function is a named section of a program that performs a specific task. Since they are not part of a class, they do not use self.	def is_five(number): return number == 5

if	Decision making is required when we want to execute a code only if a certain condition is satisfied. An if statement is a yes or no (True/False) question. 'if' is followed by instructions on what should be done when the answer is yes (True). The instructions are indented with a tab or four spaces. Conditional statements are used to perform different actions based on different conditions. The instructions that should be executed when the condition is met must be indented. An if statement consists of a Boolean expression (e.g. in) followed by one or more executional statements.	if "P" in "Python":
		send_message("P is in!!")
		if "W" in "Python":
		send_message("W is in!!")
		# "P is in!"
		# is the only message displayed

Glossary

Term	Description	Example
Import	Adding built-in Python existing libraries to the project in order to use methods and functions available within the module.	import string import random
in	'in' is is a membership operator in Python. It is used to test whether a value or variable is found in a string (or any other sequence - out of scope for this course). The operator returns 'True' if the value is found in the string and 'False' is the value is not found in the string.	"P" in "Python" # this will return True "a" in "Python" # this will return False
Initialization	Initialization is the act of assigning the first value to a variable.	my_string = "" [empty string] my_text = "Hello world"
Input	Refers to data flowing into the computer.	Typed text, files or mouse clicks
int()	Gets a string and if it represents a number, it will return its value as a number.	my_string = "4" my_number = int(my_string) - the numeric value of my_number will be 4

Integer	A numeric data type.	my_var_1 = 5 # integer
		my_var_2 = "5" # string

isalpaha()	isalpha() returns 'True' if a string includes only alphabetical characters and 'False' otherwise.	my_var1 = "Python" send_message(my_var1.isalpha()) # True
Iteration	Each running of the code/statements within the loop.	

Glossary

Term	Description	Example
len()	A function used to find out the length of a string and return its length. The string parameter can be: an explicit string or a string variable. The len() function returns the number of items (length) of an object (string, bytes, tuple, list, or range).	len("Python") will return 6

List	List is an ordered sequence of items. It is one of the most used datatypes in Python and is very flexible. All of the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].	my_list = [] # an empty list
		my_list = ["a", "b"] # a list of two elements

Method	A method is a named section of a program that performs a specific task and is written within a class.	def tick(self):
		self.minutes += 1
		if self.minutes > 59:
		self.minutes = 0
		self.hours += 1
		if self.hours > 23:
		self.hours = 0

Not	'not' is a membership operator that is used to reverse a Boolean value. not True --> False not False --> True	def is_not_five(number): return not number == 5 send_message(is_not_five(3)) # True send_message(is_not_five(5)) # False
Not in	in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence. The operator not in returns: True if the string on the left is not in the string on the right False otherwise. Not in (membership operator) - Returns True if a sequence with the specified value is not present in the object	send_message("a" not in "Python") # True send_message("a" in "Python") # False
Or	'or' is true if either of the operands is true (x or y)	x = 5 (x > 0 and x < 10) # True (x > 0 and x > 10) # True (x <0 and x > 10) # False
Output	Refers to data coming out of the computer.	Visual, auditory or tactile information
Properties	Properties of a class are a set of characteristics that describe the objects of the class.	

Glossary

Term	Description	Example
randint(low, high)	Generates a random number between low and high. To use it, import the module random each time for a different result.	import random number = random.randint(0, 4) send_message(number)
Range()	The default use of range must include a single parameter which will be at the end position of the loop. It returns a sequence of numbers and can be used to repeat a set of code a specified number of times. range() can accept three parameters: start (optional) end step (optional)	range(4) will return the following sequence - {0,1,2,3}
read_message()	This is a CodeMonkey-specific instruction that is used for accepting the user's response to the chatbot. Formal Python has other input/output instructions. The chatbot operations are halted until the user enters their response.	send_message("What is your name?") read_message()
remove()	The remove() method removes the element which is passed as an argument.	my_list = ["coding", "is", "not", "fun"] my_list.remove("not") send_message(my_list) # coding,is,fun
Return	In computer programming, a method can return a value or other information coming back from the method. Methods can calculate values and return the values in order to be used outside the method. If a return statement is not specified, the value None is returned.	def is_five(number): return number == 5

Self	The self parameter is a reference to the class instance itself and is used to access variables that belong to the class. The first parameter of any class method is always the object it is operating on, which is written in the method's parenthesis as self.	class Clock:
		  def __init__(self):
		  self.hours = 12
		  self.minutes = 30

send_message()	This is a CodeMonkey-specific instruction that is used for sending messages from the chatbot. Formal Python has other input/output instructions. Messages are displayed in the chat screen. The text to display goes inside the quotation marks. You can use variables instead of text. The variables will be used without quotations.	send_message("Hello friends") #Hello friends - is displayed in the chat screen

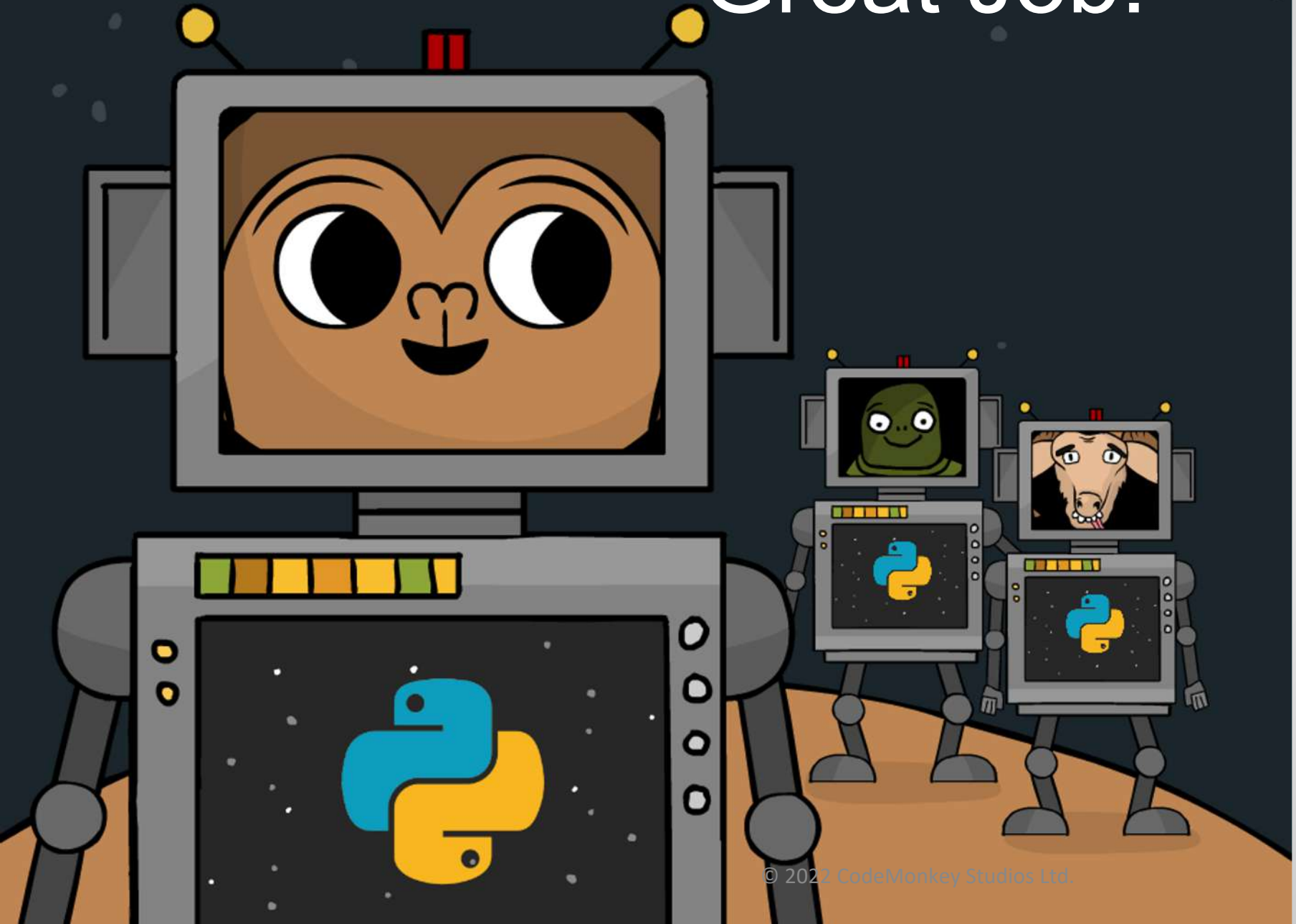
Glossary

Term	Description	Example
String	A sequence of characters. Strings are like sentences. They are formed by a sequence of characters. Variable can hold a string, A text surrounded by quotation marks is a string as well. + - Strings can be concatenated, i.e. two strings are added together into one string. The operator + concatenates two strings.	language = "Python" //language is a string variable
str()	The function str() gets a number value as a parameter and returns its representation as a string. An if statement consists of a Boolean expression (e.g. in) followed by one or more executional statements	my_var = 100 send_message("Passes - " + str(my_var) + "%!") - # "Passes - 100%!"
Test	Test is a CodeMonkey-specific compiler that is used to check compliance with instructions and Python code. Test is an internal checking mechanism that causes the messages to be sent and when a read_message is reached in the code, an automatic response is given to the Chatbot. The user does not need to enter anything. Use Test mode to check the code compliance with the game instructions.	Make sure your follow the instructions precisely before you hit test
Unicode	Unicode is a character encoding standard that has widespread acceptance.	The index of the ❤️ symbol in the Unicode table is 2764. It should be used like this:u"\u2764"

Variable	Contains one value at a time. The variable has a unique name and its value can be changed. It can be referred to by a name you give it. Variable hold values such as numbers or text (string). The value in it can be changed . Variables allow for a lot of freedom in programming. Instead of having to type out a phrase many times or remember an obscure number, computer scientists can use variables to reference them: a container (placeholder) holding one ”value” at a time. Variable is a storage location which contains a value. Each variable is identified by a symbolic name and a memory address. This separation of name and content allows the name to be used independently of the exact information it represents.	score = 100
		#score is a variable, holding a numeric value of 100
		send_message(score)
		#100 is displayed

While Loop	A while loop continues as long as its condition is met. When the condition is no longer met, the instructions after the while loop are then executed.	my_var = 3 while my_var != 0: send_message("my_var = " + str(my_var)) my_var -= 1

Great Job!



You have Completed

CODING CHATBOTS