1. Introduction and print statements

1.1. Introduction to Python

Welcome to the Code Avengers Intro to Python! In this course you will get a 5 lesson taster of what programming in Python is like.

You will learn about input and output, how to make your program make decisions, using loops to repeat code, and have bit of fun drawing graphics with Python Turtles.

Let's get into it! First, we're going to learn how to print stuff out. A print statement looks like this: `print("Hello, world!")`

This code prints the message "Hello, world!" in the console.

1. Write a print statement in the code editor that says "Hello, [your name]!" e.g. "Hello, Monty!"
2. Click to test your code.
3. Click to see if you have passed the task.

Note: When typing this code you need to type accurately; your code won't work if you leave out the `( )` or `" "`.

1.2. Printing on more than one line

If we want to print more than one line at a time, there are 2 ways we can do it. We can use **triple quotes** around the print statement and just hit enter wherever we want a new line:
```
print("""This is a
simple poem
Isn't it nice?""")
```

Or we can use a special character called a **newline** character: `\n` which will create a new line anywhere you put it in your text:
```
print("This is a \nSimple poem \nIsn't it nice?")
```

Both of these will print the same thing.

In the editor are 2 `print` statements.

1. Click to see what happens with the code how it is now.
2. Put triple quotes around the print statement on lines 1-3 so that it will print with each item on a new line.
3. Add **newline** characters `\n` into the `print` statement on line 5 so that the output is the same as the previous print statement.
4. Click to see if you have passed the task.

1.3. Printing math

We can also print numbers and math using `print` statements:
```
print(7)
print(1 + 1)
```

When we print numbers, we still need the brackets `( )` but we don't need the **quotes** `" "` - they are just for text, or what we call **strings**.

Let's try it!

1. Click to see what the code in the editor prints out.
2. On line 2, change the `print` statement so that it uses the **addition operator** and prints out `10`.
3. On line 3, write a `print` statement that uses the **divided by operator**: `/`
4. On line 4, add a `print` statement that uses **multiplication:** `*`
5. Make sure you click at the end of each task to move on!

1.4. String tricks

We can use some of the symbols we have for math to do some tricks with **strings** in print statements:
```
print("Hello" + "Bob")
print("Hello" * 5)
```

As you can see, we can **add** strings and **multiply** them.

We've put this code in the editor for you.

1. Click to see what the code does.
2. Edit the first print statement so that there's a space between the words.
3. Write another print statement using `*` that is **your name 10 times**.

1.5. Review quiz!

Let's review what you've learned in this first Python lesson!

Review Quiz Questions:

1. Which of these is the Python command that outputs text to the console?
2. What is the correct symbol for multiplication in Python?
3. Which of these print statements will print out the number 7?
4. Which of these is the special character for putting text onto a new line?

2. Variables and input

2.1. Introduction to variables

We can make our code a lot more fun by asking the user questions and using their answers. To do this we use an `input` statement and a thing called a **variable** to store their response. A variable is like a named 'container' that stores a value.

The following code asks the user's age and **stores it** (using `=`) in a **variable** called age: `age = input("What is your age?")`

We can print variables just like **strings** and numbers: `print(age)`

We don't need `" "` around a variable name either.

1. Write a line of code that asks the user's name and stores it in a variable called `name`.
2. Display the user's name using the `print` command.

2.2. Combining variables in a print statement

Another way that we can combine values such as variables with a **string** in a print statement is using a function called `format()`: `name = input("What is your name?")`
```
age = input("How old are you?")

print("{} is a nice age to be, {}!".format(age, name))
```

The values in the `format()` function are inserted in the `{}` placeholders in the order that they're written.

This is useful when we have more than one value to add in, or if the place we want to insert a value is in the middle of the string.

Add the following under the `name` input:

1. Ask the user for their favorite color and store as `color` .
2. Ask the user for their favorite food and store as `food`.
3. Print the 3 answers out in the format "Well Sharon, I bet you'd love green pizza then!"

2.3. Asking for numbers

Sometimes when we ask a question the user's answer is a number.

We might want to use their number **as a number**. For example, we might want to add to it or see if it's higher or lower than another number.

In this case we have to use a function called `int()` around our `input()` when we get their response: `age = int(input("How old are you?"))`

```
age_in_10 = age + 10
print("In 10 years you will be {}".format(age_in_10))
```

This turns the input from a **string** into an **integer**, which is a fancy word for a whole number. It's very important to make sure you have closed both sets of brackets!

1. In the editor, delete the old code and ask the user "How many brothers and sisters do you have?" and store the answer in a variable called `siblings`.
2. On the next line, add 1 to siblings and store the result in a variable called `total_children`.
3. Then, print out `total_children` in the sentence "That means your parents have {} children in total".

2.4. Variable variables

Variables are called that because their value can **vary** or **change** throughout a program.

We can change number variables by doing math calculations with them: `number = 10`

```
number = number + 5 #Now number is 15
```

There are some nice shorthand **operators** we can use to easily do the math and then store the value back in the same variable: `number = 10`

```
number += 5 # Adds 5, number = 15
number -= 5 # Subtracts 5, number = 10 again
number *= 5 # Multiplies by 5, number = 50
number /= 5 # Divides by 5, number = 10 again
```

1. What is the value of `number` on line 1?
2. What is the value of `number` on line 2?
3. What is the value of `number` on line 3?
4. What is the value of `number` on line 4?
5. What is the value of `number` on line 5?
6. Print `number` on line 6 to see if you were right about the last question and click when you are done.

2.5. Review quiz

Let's review user input and variables!

Review Quiz Questions:

1. Which of these is the best description of a variable?
2. The word "Sally" is saved in a variable called `name`. Which of these print statements will correctly print out: "That snake is named Sally"
3. True or False? An integer is a whole number.
4. Which of the following input statements will correctly take in a number that can be used in a calculation?

3. If/else statements

3.1. Making decisions

So far we have seen code that runs in **sequence**. However, often in a computer program **decisions** need to be made.

This is called **selection** in programming - making **decisions** about what should happen based on **conditions**.

This is like, if someone's parents say: "we'll get you that new phone you want IF you tidy your room." If the **condition** of tidying their room is **True** then they will get the phone, otherwise they won't.

In programming this is done with an **if statement**, which you can see in the editor. We can use any of the following **operators** to check things in a condition: `== is equal to`

```
< less than
> greather than
<= less than or equal to
>= greater than or equal to
!= not equal to
```

But for now we will focus on `==`.

1. Click and type in **pizza** all lower case to see what happens.
2. Click and type in anything other than **pizza** to see what happens.
3. Click when you're done.

Click here for an explanation of this code.

3.2. Having 2 options

It would be useful if our program also said something if the user types in a different food. It would be very time consuming if we wrote if statements for every food we could think of! Luckily, Python has an `else` branch that will help us.

The `else` branch says if it's **anything** other than the thing we checked for in the `if`, then do this e.g.

```
if fave_food == "pizza":
  print("Yum!")
else:
  print("Yuck")
```

In this code, if you type `"pizza"` it will say `"Yum!"` and if you type **anything else at all** it will say `"Yuck"`. This is why `else` doesn't need a **condition**.

1. On line 5 in the editor, add the `else:` keyword.
2. Inside the `else` branch, print "Yuck".
3. Click and test it with both **pizza** and another food.

Note: For code to be **inside a branch** it has to be **indented** or tabbed in one tab. Make sure your code is laid out like the example with the print statements lined up correctly.

3.3. Creating quiz questions

Let's practice using `if`/`else` structures by creating a small quiz.

In the quiz, we will ask the user a question, then check if the answer is correct. If it is, we'll tell them that, otherwise we'll tell them they were wrong.

1. In the editor, ask the user "What does CPU stand for?" and store their response as `answer`.
2. On the next line, write an **if statement** to check if their answer is "central processing unit".
3. If it is, inside the `if` branch, print "Correct!"
4. Add an `else` branch.
5. Inside the else branch print "Sorry, wrong answer."
6. Repeat this block of code to ask and check the 2 questions below.
7. Click and test the quiz out.

Q2: "How many bits are in a byte?"

A: **8**

Q3: "Which is bigger: a kilobyte or a megabyte?"

A: **megabyte**

3.4. Adding a score variable

This quiz would be more interesting if it told us how many questions we got right!

Inside an `if` or `else` branch, you can have as many lines of code as you want, as long as they are all **indented** correctly.

Let's add in a `score` variable and give the user 1 point for each correct answer.

1. On line 1 create a variable called `score` and set it equal to `0`.
2. Inside the first **if statement** on line 6 add one point to `score` after printing "Correct!"
3. Add 1 point to `score` if they get the second question correct.
4. Add 1 point to `score` if they get the third question correct.
5. At the end, print their score in the format "You scored {} points!" using `.format()` and the `score` variable.
6. Click and test the quiz a few times.

3.5. Review quiz

Let's review **selection/conditional** code!

Review Quiz Questions:

1. Which character goes at the end of a line of code that starts with `if`?
2. What is the keyword used for making a second branch in an if statement that will run whenever the if condition is False?
3. Which operator means **is equal to** in Python?
4. How do you know which bit of code is inside an if statement?

4. Using loops to repeat code

4.1. Repeating code

Sometimes we want to do something more than once in a row in a program. We could write the code several times, but we also have structures that are also known as **loops**. Loops repeat code for us.

We will look at 2 types of loops: **for loops** and **while loops**.

Let's take a look at our first loop... the `for` loop: `for i in range(5):`
`  print(i)`

A **for loop** repeats the **block** of code inside it a **set number** of times. `i` is the built-in **counter** that keeps track of how many times the loop has repeated.

1. Run the code in the editor to see how `i` changes as the loop is run.
2. Change the loop statement so that the numbers **0-9** are printed.

Note that because it **starts from 0**, the last number printed is always **one less** than the number we put in `range()`.

4.2. Customise a `for` loop

We can customise a `for` loop in several ways including making it start and stop at **different numbers.** We can also print anything inside the loop, not just the numbers themselves. For example: `for i in range(5):`
`  print("Hello!")`

This will print "Hello!" 5 times on separate lines.

1. Click to see what the code in the editor does.
2. Compare the output to the code and see if you can figure out how it works. Make sure you don't change these 3 loops.
3. Add a for loop to line 18 that prints out the numbers **7**, **8** and **9**.
4. Click to see if you got it right!

4.3. Meet the `while` loop

The `while` loop can be used in a similar way to a `for` loop for printing out numbers, but there are some differences.

In a `while` loop, instead of saying how many times we want the loop to repeat, we use a **condition** -- just like in an `if` statement. The loop will run until the **condition is met**: `while i <= 5:`
`  print(i)`

This is the basic structure of a `while` loop, but we have to write a bit more code for it to work because it doesn't have a **built-in counter** like the `for` loop.

1. Look at the code in the editor and see what we have to do with `i` to make it work.
2. Change the code so that the numbers **0-10** are printed.
3. See if you can change the code so that it **starts from 1**.
4. **(Optional)** Change the conditional **operator** so that **10** is **NOT** printed.

So we have to set up the counter first, with the start point, and then **increment** it inside the loop.

4.4. Use other conditions in a `while` loop

Because a **while loop** runs on a **condition**, we don't have to use just numbers. We can write any condition we want in there, but be careful because something like: `while 3 < 4` will make an **infinite loop**, because it will never be **False**, and so won't get switched off!

Have a look at the code editor to see a condition based on the user's input.

1. Click to see what the code does. Type "no" a few times. Then type "yes".
2. Change the condition in the `while` statement so that the loop repeats as long as **response is NOT equal to "yes"**.
3. Click again and test it the same way as in step 1 - it should work the same way.

Reminder:

```
== is equal to
< less than
> greather than
<= less than or equal to
>= greater than or equal to
!= not equal to
```

Code explanation: Although both versions of this loop have the same results for **yes** and **no**, in some cases there's a good reason to write the condition one way instead of the other. For example, if we are checking whether `response == "no"`, typing "not yet" will stop the loop when we want it to keep going. If we check for `response != "yes"`, however, the loop will continue when we type "not yet".

Click here for an example.

4.5. Review quiz

Let's review loops!

Review Quiz Questions:

1. How many times will "Ni!" be printed out with the following loop?
2. What is the last number that the following loop will print out?
3. How many times will "Hello!" be printed out with the following loop?
4. What is the last number that the following loop will print out?

5. Drawing graphics with Turtles

5.1. Meet Tia the Turtle

We're going to finish off this introduction by playing with some **Turtles**!

Turtle graphics are a bit of a programming tradition and they really help to understand the idea of **sequence** in programming.

The example code shows you how to create a turtle called `tom` by first **importing** the `turtle` module.

1. In the code editor `import` the `turtle` module.
2. On line 2 create a turtle called `tia`.
3. Type `tia.forward(50)` on line 3.
4. Click to see what happens.

5.2. Move and turn Tia

We can **move** a turtle forwards or backwards using:
```
tom.forward(50)
tom.backward(100)
```

The number in the brackets is how many **pixels** (px), or how far, the turtle will move.

We can also **turn** a turtle using: `tom.left(90)`
`tom.right(45)`

Here, the number is the **angle** it will turn, in degrees. So `90` will make it turn a right angle, `180` will turn it to face the opposite direction and `360` will turn it all the way around.

1. Click to see what the code in the editor does.
2. Complete the code to make `tia` draw a **square**.
3. Modify the code so that `tia` draws a square that is **200px x 200px**.

5.3. Change the color and size

Great! So we're drawing with our turtle. We can **change the color and size** of the line that gets drawn using: `tom.color("red")`
`tom.pensize(5)`

1. On line 3 set the `pensize()` to `10`.
2. Edit your code for the square so that each side is a different color, in the order: **red, green, yellow, blue**.
3. Click to see the image!

These commands happen in **sequence** or we can say they are **sequential**.

It's very important in programming to be able to break down a problem into **steps that happen in order**. And then we have to **make sure** that those things **do** happen in that exact order!

If you change the color or turn in the wrong place, you will end up with a different result.

5.4. Draw some different shapes

You can draw all kinds of things with turtles, but you have to think carefully about how to do it. The code for our square was pretty repetitive in the last task, so let's try it with loops instead!

To use loops with turtle graphics, you have to think carefully about which steps need to be repeated. To draw a regular shape, you will need the line of code that **moves the turtle forward**, and the line that **turns it the correct angle.**

NOTE: Use `.left()` for all of the shapes in this exercise.

1. On line 3 set the `pensize()` to `5`.
2. On line 6 replace the `???` to set the color to **purple**.
3. We've added the for loop statement, so inside the loop write the 2 lines of code you need to make a **200px x 200px square**.
4. Under the last comment write the code to draw an **orange triangle** with sides that are **200px.**

Hint for triangle angles.

5.5. Filling shapes with colors

For the last task in this intro course, we're going to **fill shapes** with a color. There are 3 commands we need for this:

`tia.fillcolor()` - sets the color for filling.

`tia.begin_fill()` - goes just before the line of code that starts drawing your shape.

`tia.end_fill()` - goes right after the last line of code used to draw the shape.

1. Set the pen size on line 5 to `8`.
2. Set the `fillcolor()` to **yellow** on line 8.
3. Add `begin_fill()` on line 9 - we don't need anything in these brackets.
4. Add `end_fill()` on line 13.
5. Click to see the result.

Congratulations on completing this **Introduction to Python**! If you enjoyed it you might like to extend your skills even further in our Level 1 Python Course.