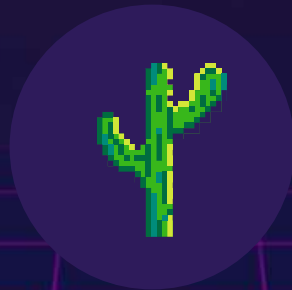


# 2.3 PLAYING WITH LOOPS



keep playing on repeat





## 2.3 PLAYING WITH LOOPS

- It is very common to have repetitive elements in games.
- These elements can be coded once but run many times.
- Looping structures enable us to run the same block of code over and over again.
- We will learn two types of loops: "for" and "while".





# FOR LOOPS

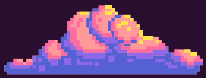
A "for loop" is made up of a loop variable and a loop body.

The loop variable keeps track of the number of repetitions that occurred.

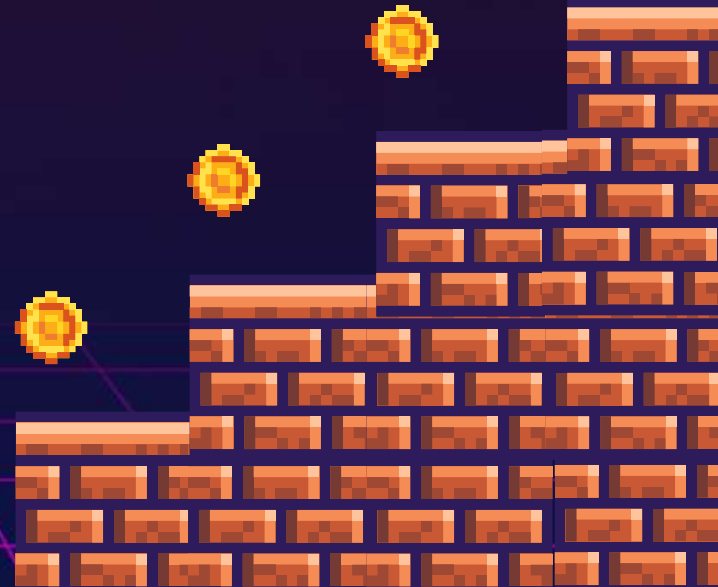
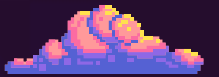
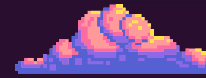
The block of code that gets repeated in a loop is the loop body.

block





# FOR LOOPS



1. 2. 3. 4.



**loop variable**

for count in range(1, 5):  
print(count)

**loop body**

Repeated code. Always indent commands inside a loop.

Keeps track of how many times the loop has run so far.

1  
2  
3  
4



# RANGE( ) FUNCTION

range

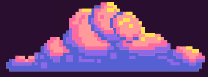
```
for count in range(1, 5):  
    print(count)
```

The range function gives us a list of numbers from the first number up until the second-to-last number. Therefore, `range(1, 5)` gives us a list of `[1, 2, 3, 4]`. Note that 5 is not included.

# WHAT DOES THIS DO?

```
guesses = []  
print("Guess three numbers...")  
for count in range(1, 4):  
    guess = input("Enter...")  
    guesses.append(guess)
```

- A. Loops through numbers 1 up to 3.
- B. Conditional branch to select code.
- C. Loops through numbers 1 up to 4.
- D. Stores multiple user inputs in a list.



# Did you understand?



## COMPLETE THE PROGRAM

Write a program that displays all the prime numbers between 2 and 10.

 Fill in the Blanks

```
_____ n in _____(2, 11):
```

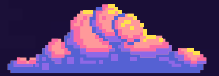
```
_____ n == 2 or n == 3:
```

```
    print(n)
```

```
else:
```

```
    prime = (6 * n) - 1
```

```
    print(prime)
```



range

if

for





# LOOPING OVER A LIST

```
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print("Eat this " + fruit)
```

**fruit var**

Fruit is a looping variable that moves along the list and stores the current fruit.

FRUITs = [...]

frUIT IN FRUITs?

true

DISPLAY  
EAT FRUIT





# LOOPING OVER TWO LISTS

```
fruits = ["apple", "banana", "cherry"]
names = ["John", "Paul", "Amy"]
index = 0
for fruit in fruits:
    print(names[index] + " eats a " + fruit)
    index = index + 1
```

**index**

We created an index variable which moves through the fruit list.

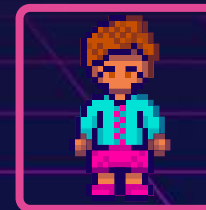
0



1



2





# TWO LIES AND A TRUTH

Can you pick out any lies?

```
fruits = ["apple", "banana", "cherry"]
names = ["John", "Paul", "Amy"]
index = 0
for fruit in fruits:
    print(names[index] + " eats a " + fruit)
    index = index + 1
```

★ Multiple Choice

A. Amy eats a cherry

B. John eats a banana

C. Jake eats an apple





# FOR LOOP SUMMARY

Loop through consecutive numbers by making use of the `range` function.

When looping through a list the loopin `variable` stores the current item.

Create a variable to track the current `index` when looping over two lists.

`number`





# LESSON CHALLENGE

- Time to put the theory into practice.
- You will continue to build a small component of a word search game.
- You have to use all you learned so far.
- Find your tasks!





# WHILE LOOPS

A "while" loop uses a loop condition which can be True or \_\_\_\_\_.

The loop \_\_\_\_\_ determines whether the loop body should be run.

The loop body will only run when the loop condition evaluates to \_\_\_\_\_.

condition

false

repeated

True

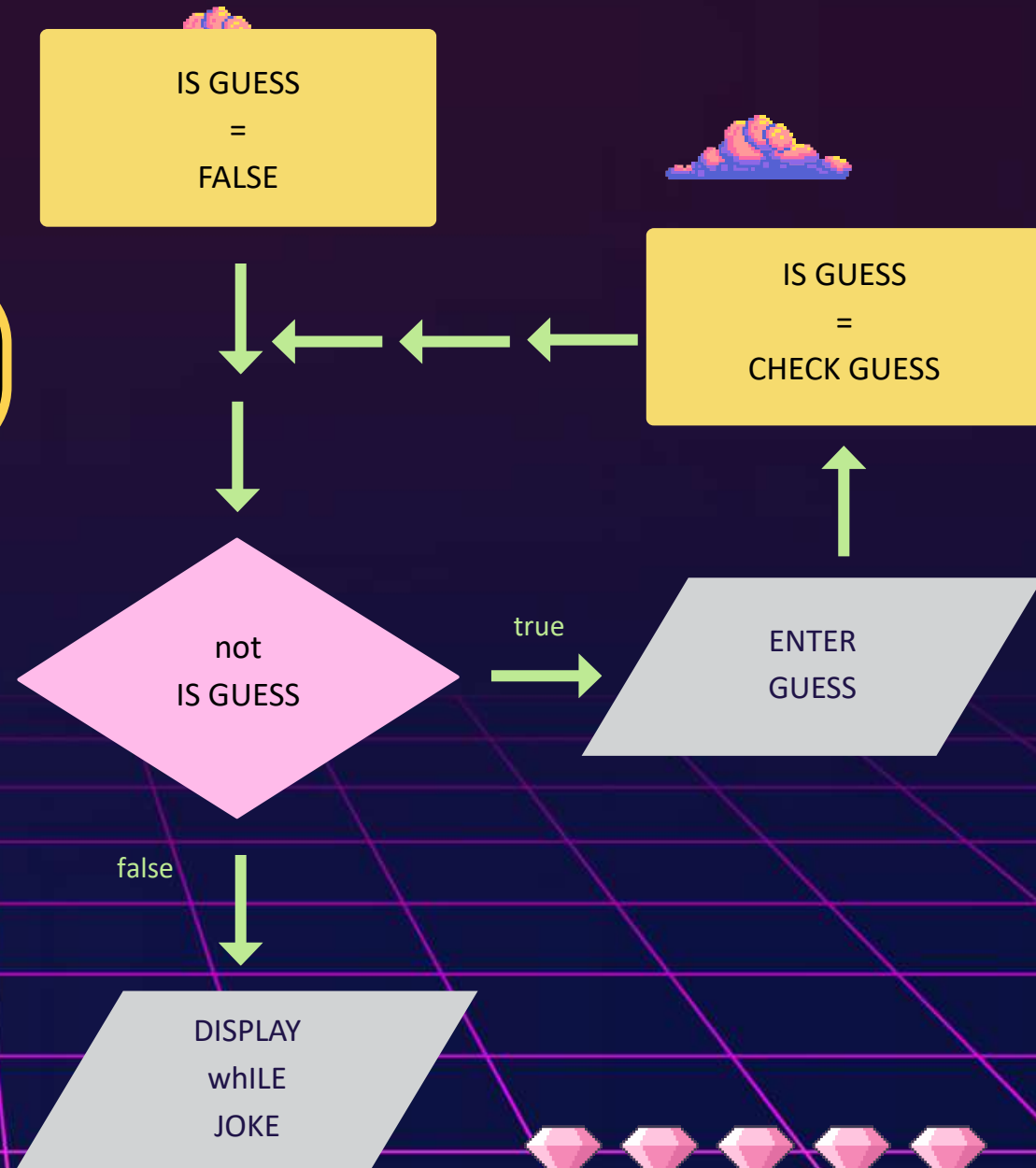


# WHILE LOOP

```
is_guess = False
while not is_guess:
    guess = input("Enter guess: ")
    is_guess = check(guess)
    print("Took a while to guess!")
```

loop condition

not False equates  
to  
True!



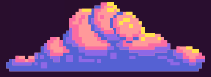


# WHAT IS THE DIFFERENCE?

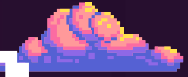
- When we know exactly how many times we need a loop to repeat we should use a **for loop**.
- At times, we might not know exactly how many times we need a loop to repeat, and so we use a **while loop**.
- A "for" loop has a clear start and end like a flight of stairs.
- A "while" loop will only start if the loop condition is True and will stop when the loop condition is False.







# DID YOU UNDERSTAND?



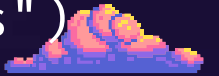
## COMPLETE THE PROGRAM

Complete the code snippet that will be part of a game called "Guess the number". The program keeps asking the user to guess the number when the user input is incorrect.

```
is_correct = False
```

```
_____ not is_correct:
```

```
    guess = input("Enter your guess")
```



```
    _____ guess == number:
```

```
        is_correct = True
```

```
    else:
```

```
        _____("Try again.")
```

print

if

while



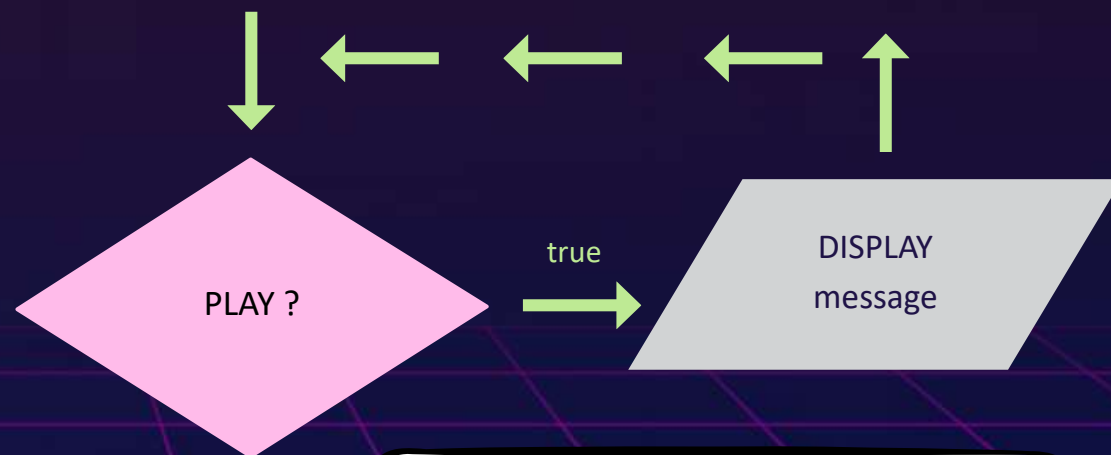
# TO INFINITY...

```
play = True
while play:
    print("Will this ever end?")
```

**infinite loop**

This loop will run forever! Use with extreme caution.

play  
=  
TRUE



You may have done this unintentionally. It happens. When your program has an infinite you can press Ctrl + C to stop.

# MAKE A BREAK FOR IT!

```
lives = 5
while True:
    lives_lost = game_cycle()
    lives = lives - lives_lost
    if lives == 0:
        break
```

**break**

A command that  
enables you to terminate  
a loop

Might seem strange, but some programmers intentionally put infinite loops. However this is dangerous and if you want to do this make sure there is a point where you "break" from it.



# LESSON CHALLENGE

- Time to put the theory into practice.
- You will continue to build a small component of a word search game.
- You have to use all you learned so far.
- Find your tasks!

