



## Match Me

Have you ever played a game that tests your memory? What types of memory games have you played? Did they involve colors, objects, or sounds? The Match Me game generates a list of random colors the player must match. Originally, the list has only one color. If the player answers the question correctly, the game adds another random color to the growing list of colors and progresses to the next level. Game play continues while the player recalls the ever-growing list of colors.

### Objectives:

#### Programming Objectives:

- Write procedures to manage abstractions.
- Write selection statements.
- Write iterative statements.
- Generate random integer values to represent colors.
- Use lists to manage abstractions.

### Key AP Computer Science Principles Standards:

- Represent a value with a variable (AAP-1.A)
- Represent a list or string using a variable (AAP-1.C)
- Write conditional Statements (AAP-2.H)
- Write nested conditionals (AAP-2.I)
- Select appropriate libraries or existing code segments to use in creating new programs (AAP-3.D)
- Write iteration statements (AAP-2.K)
- Write expressions that use list indexing and list procedures. (AAP-2.N)
- Write iteration statements to traverse a list. (AAP-2.O)
- Write statements to call procedures (AAP-3.A)
- Develop procedural abstractions to manage complexity in a program by writing procedures. (AAP-3.C)
- For generating random values, write expressions to generate possible values. (AAP-3.E)

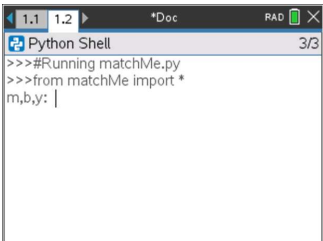
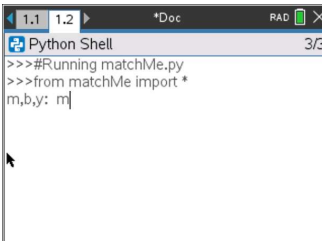
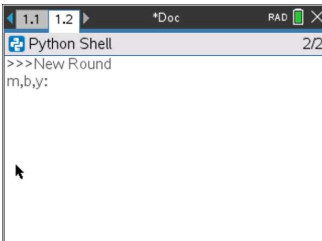
### Game Set-Up:

To play the game, you will need either a TI-Nspire CX II calculator or a Computer with TI-Nspire CX II software.



Connect your TI-Nspire CXII calculator or computer to the TI-Innovator Hub.

Let's simulate a quick game of "Match Me".

 <p>Game generates a random color. It displays a magenta light</p>	 <p>User correctly enters an m.</p>	 <p>Since m was correct, game play continues. A new color is generated and added to the list. This time it displays a magenta light, then a blue light</p> <p>Game play continues until the user enters an incorrect sequence.</p>
---	--	--

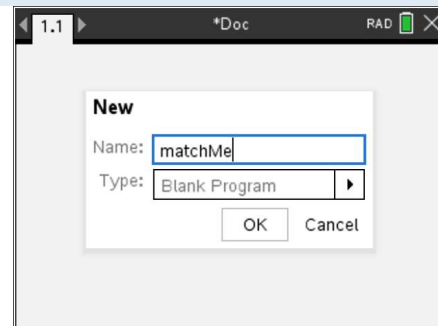


Coding Process:

1. Create a new program named “**matchMe**”.

Choose Data Sharing for the default type.

\*You can name your project “matchMe” all in lower case. However, using capital letters for the M makes it easier to read the name of the file. Often, programmers capitalize the first letter of each new word in a variable name. This form of naming is referred to as “camel case”. It doesn’t change how the program runs; it does however make it easier for the programmer to read.



You need 3 different libraries for this project.

- a.) The library that has the hub functions.
- b.) The library that contains functions for generating random integers.
- c.) The library that lets you clear the screen.

Import the three libraries.

2. The basic matching game requires a few basic ideas.

- \*The computer generates a list for the user to match.
- \*The user enters a list the same length as the computer’s list.
- \*The computer compares the two lists.
  - If they match, continue game play adding new items to the list,
  - otherwise exit game play.

3. Now that you know the basics, let’s create a more detailed look at the game.

- \*Create a list of possible colors for the list.
- \*Create a list of tones to go with each color.
- \*Create the initial computer list.
- \*Create an empty list for the user.
- \*Make sure the hub’s light is turned off.

- \*Create a loop that will continue while the user is “winning”:
  - Display the computer’s list

Ask for and store the user’s list. This needs to be the same length as the computer’s list.

Play back the list the user entered using the appropriate colors and tones.  
While playing back the list, determine if the user is correct.

If the user is correct, clear out the screen history and add a new item to the list.

- \*After the user enters a wrong answer and loop has been exited, tell the user how many levels were passed.
- \*Display both the computer’s and the user’s list so the user can see the mistake.



4. Let's start coding!

You will need three different colors for your game.  
(You may choose more if you like.)

What colors will you use? For example: color.rgb(248, 0, 255) is magenta.

Determine your colors:

color.rgb(\_\_\_\_, \_\_\_\_, \_\_\_\_)  
          red  green  blue

color.rgb(\_\_\_\_, \_\_\_\_, \_\_\_\_)  
          red  green  blue

color.rgb(\_\_\_\_, \_\_\_\_, \_\_\_\_)  
          red  green  blue

Now that you have your colors, you will put them in three lists one for red, one for green and one for blue. Put all the red elements in the list r, green elements in g and blue elements in b.

r = [\_\_\_\_, \_\_\_\_, \_\_\_\_]

g = [\_\_\_\_, \_\_\_\_, \_\_\_\_]

b = [\_\_\_\_, \_\_\_\_, \_\_\_\_]

When you code these three lists, add a comment with the key. List your colors and their location. For example, if magenta is my first color, I would put magenta-0. The comment will remind you the order in which you added colors. This will be useful for debugging later.

5. Next determine 3 different tones to play, one for each color.  
Pick values between 200 and 400.

t = [\_\_\_\_, \_\_\_\_, \_\_\_\_]

6. Create a list with one random initial value from 0-2. This will be the first color displayed in the computer's list.

lst = [randint(0,2)]

7. Create an empty list to hold the player's values.

player = [ ]

```

1.1 *Doc RAD 5/11
*matchMe.py
from ti_hub import *
from random import *
from ti_system import *

#keys
r = [ ]
g = [ ]
b = [ ]

```

```

1.1 *Doc RAD 10/11
*matchMe.py
from ti_hub import *
from random import *
from ti_system import *

#keys
r = [ ]
g = [ ]
b = [ ]
t = [ ]

```



8. Turn off the hub light.

```
color.off()
```

9. It's a good idea to test your code as you go. Let's add a light to display the current color for the computer.

This value is in item 0 for the r, g and b lists. Type:

```
color.rgb(r[lst[0]], g[lst[0]],b[lst[0]])
```

Execute your code several times. Verify you get all three colors you wanted.

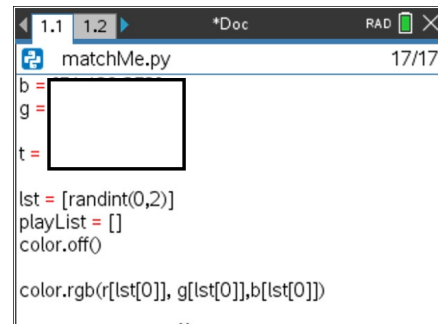
It might also be beneficial to change the color.rgb line to:

```
color.rgb(r[0], g[0],b[0])
```

Make sure the 0 item is the color you think it is.

Then try: color.rgb(r[1], g[1],b[1]) to make sure you know which color matches index 1.

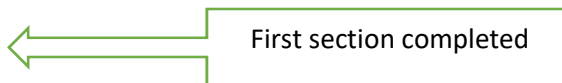
When you are done testing your code, remove the color.rgb line.



10. Execute your code. Make sure you don't have any errors.  
If you do, fix them before you continue.

11. Look back over our project outline:

~~Create a list of possible colors for the list.~~  
~~Create a list of tones to go with each color.~~  
~~Create the initial computer list.~~  
~~Create an empty list for the user.~~  
~~Make sure the hub's light is turned off.~~



Create a loop that will continue while the user is "winning":  
Display the computer's list

Ask for and store the user's list. This needs to be the same length as the computer's list.

Play back the list the user entered using the appropriate colors and tones.

While playing back the list, determine if the user is correct.

If the user is correct, clear the screen history and add a new item to the list.

After the user enters a wrong answer and loop has been exited, tell the user how many levels were passed.  
Display both the computer's and the user's list so the user can see the mistake.



12. Inside the loop, the first code segment says it should display the computer's list. To modularize your code, we will create a function. The function will take a list as input, then loop through the list and display the appropriate tone and color for each item in the list. Along with passing the color list into the function, you will pass in your r, g, b and t lists.

13. Scroll to the top of your program. Create a definition named `display` that takes five list parameters named `lst`, `r`, `g`, `b` and `t`.

Use a for loop to cycle through each item in the list `lst`.  
Remember the `len()` function will find the length of your list.  
As you go through the list, store the number from the list in a variable named `p`.

```
*matchMe.py 6/21
from ti_hub import *
from random import *
from ti_system import *

def display(lst,r,g,b,t):
    for i in range( ):
        p=lst[i]
    
```

14. Add lines of code that will:

set the rgb color using the lists `r`, `g` and `b` as well as the index `p`.  
set the tone to tone `p` in the list `t`.  
pause for  $\frac{1}{2}$  a second.

```
*matchMe.py 6/24
from ti_hub import *
from random import *
from ti_system import *

def display(lst,r,g,b,t):
    for i in range(len(lst)):
        p=lst[i]
        color.rgb(
        sound.t
        sleep(0.5)
    
```

15. Turn the light off and pause for 0.25 seconds.

```
*matchMe.py 12/26
from ti_system import *

def display(lst,r,g,b,t):
    for i in range(len(lst)):
        p=lst[i]
        color.rgb
        sound.to
        sleep(0.5)
    
```

16. Double check your display function.

Does it pass in 5 variables `lst`, `r`, `g`, `b` and `t`?

Do you have a loop that stores each item in the list to a single variable `p`?

Do you use `p` to display the appropriate rgb color using the arrays `r`, `g` and `b`?

Do you play a tone using `p`?

Do you pause for 0.5 seconds, turn the light off, then pause for 0.25 seconds?



# Computer Science with Python

## TI-NSPIRE™ CX II TECHNOLOGY

17. Did you know you can check your function display to make sure it works properly?

At the end of your code add two lines of test data:

```
lst=[0,2,0,1]
display(lst,r,g,b,t)
```

Make sure you are connected to your TI-Innovator Hub.

Execute your code. It should display 4 different colors. The first and third colors played should be the same (color 0). The second color played should be your last color, color 2.

If you had any errors when you ran your program, fix them before you continue.

## MATCH ME

### STUDENT DOCUMENT

```
*matchMe.py 28/28
g = [139,128,15]
t = [260,290,320]
lst = [randint(0,2)]
playList = []
color.off()

#test data
lst=[0,2,0,1]
display(lst,r,g,b,t)
```

18. Remove your three lines of test code.

```
*lightsColor.py 26/26
g = [139,128,15]
t = [260,290,320]
lst = [randint(0,2)]
playList = []
color.off()

|
```

19. Look back over our project outline:

~~Create a list of possible colors for the list.~~  
~~Create a list of tones to go with each color.~~  
~~Create the initial computer list.~~  
~~Create an empty list for the user.~~  
~~Make sure the hub's light is turned off.~~

First section completed

Create a loop that will continue while the user is "winning":

~~Display the computer's list~~

Function written and tested

Ask for and store the user's list. This needs to be the same length as the computer's list.

Play back the list the user entered using the appropriate colors and tones.

While playing back the list, determine if the user is correct.

If the user is correct, clear the screen history and add a new item to the list.

After the user enters a wrong answer and loop has been exited, tell the user how many levels were passed.

Display both the computer and the user's list so the user can see the mistake.



# Computer Science with Python

## TI-NSPIRE™ CX II TECHNOLOGY

20. Next, you'll write a function that will ask for and store the user's color list. You'll create a parallel list with numeric values for the colors. Both lists need to be the same length as the computer's list. This function will return the two newly created lists.

Create a function named `getUserList()` that takes a list length parameter **length**, an empty list for colors named **playC**, and an empty list for numbers named **playList**.

21. Inside the function, write a loop that will happen **length** times.

Ask the user to input one of the three given colors. For example, if your colors are magenta, blue, and yellow, you could ask for an "m", "b" or "y".

Add the user's input to the list `playerC`.

Convert the letter to a number. Look back at your code and comment from step 4. Use an `if..elif..else` statement to add the appropriate number based on the color entered to `playList`.

22. Return both the `playC` list and the `playList`.

23. Let's test your code. You've already tested your first function `display`. Now you will test your second function `getUserList`.

This function uses the length of the user's list to create the player's lists. It should return a list with string values for the user's list and another list with the numeric values 0,1 or 2 associated with those strings.

Add the lines:

```
playC=[]
print( getUserList(4, playC, playList) )
```

Execute your code. Does it request 4 inputs?  
Does it print two lists similar to the ones on the right.  
The demo code entered 2, 1, 0, 2 as the 4 values.

Fix any errors before you continue.

## MATCH ME

### STUDENT DOCUMENT

```
*matchMe.py 12/28
def display(lst,r,g,b,t):
    ...
def getUserList(length, playC, playList):
    ...
```

```
*matchMe.py 24/37
def getUserList(length, playC, playList):
    col=input()
    #add to col to playerC
    playC.append(col)
    #convert
    if col=="m":
        playL.append(0)
    elif col=="b":
        playL.append(1)
    else:
        playL.append(2)
```

```
*matchMe.py 22/38
col=input()
#add to col to playerC
playC.append(col)
#convert
if col=="m":
    playL.append(0)
elif col=="b":
    playL.append(1)
else:
    playL.append(2)
return playC, playL
```

```
*matchMe.py 38/39
b = [51,128,253]
g = [139,128,15]
t = [260,290,320]
lst = [randint(0,2)]
playList = []
color.off()
#test code
playC = []
print( getUserList(4, playC, playList) )
```



24. Remove your test code.

```

1.1 1.2 *Doc RAD 38/38
*lightsColor.py
b = [51,128,253]
g = [139,128,15]

t = [260,290,320]

lst = [randint(0,2)]
playList = []
color.off()

```

25. Look back over our project outline:

~~Create a list of possible colors for the list.~~  
~~Create a list of tones to go with each color.~~  
~~Create the initial computer list.~~  
~~Create an empty list for the user.~~  
~~Make sure the hub's light is turned off.~~

← First section completed

Create a loop that will continue while the user is "winning":

~~Display the computer's list~~

← Function written and tested

~~Ask for and store the user's list.~~

~~This needs to be the same length as the computer's list.~~

← Function written and tested

Playback the list the user entered using the appropriate colors and tones.

While playing back the list, determine if the user is correct.

If the user is correct, clear out the screen history and add a new item to the list.

After the user enters a wrong answer and loop has been exited,

tell the user how many levels were passed.

Display both the computer and the user's list so the user can see the mistake.

26. Next, you'll write a function to playback the list the user entered using the appropriate colors and tones. While playing back the list, determine if the user's list matches the computer's list.

This function should:

loop through all the items in the user's list:

play the user's color for each item in the list

play the appropriate tone for each item in the user's list

Make sure the user's list matches the computer's list. If the lists match, return 1 to continue playing. Otherwise, return 0 to exit playing.

What parameters do you need to pass to the function?

What value or values do you need to return?





# Computer Science with Python

## TI-NSPIRE™ CX II TECHNOLOGY

27. Create a function named playBackUser. Pass in the lists playList, lst, and t.

## MATCH ME

### STUDENT DOCUMENT

```

1.1 1.2 *matchMe RAD 28/42
*matchMe.py
if col=
play
elif col
play
else:
play
return

def playBackUser(playList,lst,t):

```

28. Inside the playBackUser function, create:

- a loop that will loop through all the items in the playList:
  - store the current item from the list in a variable named num
  - change the color on the TI-Innovator Hub to the rgb value for num. Use the lists, r, g and b.
  - play a tone using the list t and the num variable for 0.5 seconds.
  - pause for 0.5 seconds
  - turn the led off

```

1.1 1.2 *matchMe RAD 34/48
*matchMe.py
Previously coded
return

def playBackUser(playList,lst,t):
for i in
num
col
sou
sle
col

```

29. Lastly, add an if statement. If the user's playList matches the computer's lst, return 1 otherwise return 0.

```

1.1 1.2 *matchMe RAD 27/51
*matchMe.py
def playBackUser(playList,lst,t):
for i in
num
col
sou
sle
col
if playL
retu
else:
retu

```

30. Before you continue, let's make sure the function playBackUser works.

For the first test, match the user and computer lists.

The program should play all the user's colors and return 1 because they match.

Add the following test code to your program.

```

#test code
playList = [0,2,1,2]
lst = playList
print( playBackUser(playC, lst, t) )

```

```

1.1 1.2 *matchMe RAD 55/56
*matchMe.py
t = [260,290,320]

lst = [randint(0,2)]
playList = []
color.off()

#test code
playList = [0,2,1,2]
lst = playList
print( playBackUser(playC, lst, t) )

```

If your code has an error or doesn't work properly, fix it before you continue.



# Computer Science with Python

## TI-NSPIRE™ CX II TECHNOLOGY

31. Alter your test code. This time, the playList  $\neq$  lst.  
Therefore, it should print a 0.

```
#test code
playList = [0,2,1,2]
lst = [0,0,0,0]
print( playBackUser(playC, lst, t) )
```

If your code has an error or doesn't work properly, fix it before you continue.

32. Remove your test code.

## MATCH ME

### STUDENT DOCUMENT

```
*matchMe.py 55/56
t = [260,290,320]

lst = [randint(0,2)]
playList = []
color.off()

#test code
playList = [0,2,1,2]
lst = [0,0,0,0]
print( playBackUser(playC, lst, t) )
```

```
*matchMe.py 51/51
t = [260,290,320]

lst = [randint(0,2)]
playList = []
color.off()
```

33. Look back over our project outline:

~~Create a list of possible colors for the list.~~  
~~Create a list of tones to go with each color.~~  
~~Create the initial computer list.~~  
~~Create an empty list for the user.~~  
~~Make sure the hub's light is turned off.~~

First section completed

Create a loop that will continue while the user is "winning":

~~Display the computer's list~~

Function written and tested

~~Ask for and store the user's list.~~

~~This needs to be the same length as the computer's list.~~

Function written and tested

~~Playback the list the user entered using the appropriate colors and tones.~~

~~While playing back the list, determine if the user is correct.~~

Function written and tested

If the user is correct, clear out the screen history and add a new item to the list.

After the user enters a wrong answer and loop has been exited,

tell the user how many levels were passed.

Display both the computer and the user's list so the user can see the mistake.



34. Now you'll code the main loop that contains the function calls.

~~Create a list of possible colors for the list.~~  
~~Create a list of tones to go with each color.~~  
~~Create the initial computer list.~~  
~~Create an empty list for the user.~~  
~~Make sure the hub's light is turned off.~~

Create a loop that will continue while the user is "winning":

~~Display the computer's list~~

Function written and tested

~~Ask for and store the user's list.~~

~~This needs to be the same length as the computer's list.~~

Function written and tested

~~Playback the list the user entered using the appropriate colors and tones.~~

~~While playing back the list, determine if the user is correct.~~

Function written and tested

If the user is correct, clear out the screen history and add a new item to the list.

After the user enters a wrong answer and loop has been exited,  
tell the user how many levels were passed.

Display both the computer's and the user's list so the user can see the mistake.

35. Create a variable named win. Set it equal to 1.  
While the win variable is one, game play will continue.

```
1.1 1.2 *matchMe RAD 54/54
*matchMe.py
t = [260,290,320]
lst = [randint(0,2)]
playList = []
color.off()

win = 1
while win == 1:
  **
  **
```

36. Add a function call to your function display.

What parameters does the function use?

Does it return a value? If so, what variable does it return?



37. Add the display function call.



```
lst = [randint(0,2)]
playList = []
color.off()

win = 1
while win == 1:
    #call the display function
    display(lst,r,g,b,t)
```

38. Next, you'll add the function call for getUserList.

What parameters does this function call require?


Does this function return any variables? If so, which ones?

39. Before you can call the getUserList, you'll need to create empty lists playC and playList. You'll pass these lists in as parameters use them to catch the return values.

Add the code:

```
playC=[]
playList=[]
playC,playList=getUserList(len(lst), playC, playList)
```

\*Remember, getUserList needs the length of lst not lst.



```
win = 1
while win == 1:
    #call the display function
    display(lst)
    #call the getUserList function
    playC=[]
    playList=[]
    playC,playList=getUserList(len(lst), playC, playList)
```

40. Next, you'll add the function playBackUser.

What parameters does this function call require?

Does this function return any variables? If so, which ones?



## Computer Science with Python

### TI-NSPIRE™ CX II TECHNOLOGY

41. Add your function call.

42. You have 5 more lines to add to the while function.

If win equals 1:  
clear the history on the screen  
print the words "New Round"  
add another random integer between 0 and 2 inclusive.  
pause for 1 second so the user see's the words "New Round".

43. Execute your code.

At this point, you should be able to play a simple version of your game.

Answer the problems correctly 2 or three times in a row. Game play should continue.

If you answer incorrectly, the game should exit. Currently, the program just exits.

44. Create a variable named correct that is equal to the length of the computer's list. Use this variable to display the number of rounds answered correctly.

## MATCH ME

### STUDENT DOCUMENT

```
*matchMe.py 65/65
display(lst)
#call the getUserList function
playC=[]
playList=[]
playC,playList=getUserList(len(lst), playC, pla
#call the playBackUser function
#store results in win
```

```
*matchMe.py 70/70
#call the playBackUser function
#store results in win
win = playB previously coded
if win==1:

```

```
*matchMe.py 73/73
win = playBackUser(playList,lst,t)
if win==1:
Previously coded
correct = len(playList)
print(correct)
```



## Computer Science with Python

### TI-NSPIRE™ CX II TECHNOLOGY

45. Create a new variable userC.  
Loop through all the numbers in the list lst.  
Use an if...elif...else statement to add the colors to your userC list.

46. Print the computer's list.  
Print the user's list.

47. Run you code several times.

Fix any errors.

48. Optional Challenges:

- 1.) The code in steps 44-47 could be made into a function named finalDisplay. What parameters would this function need? Would it return any variables? Create this function, then replaced the code at the bottom of your program with a call to this function.
- 2.) Add a 4<sup>th</sup> color to your game. Modify the code in your game to run properly with this added color.
- 3.) Create a scoring system. The system should be more complicated than simply adding 1 point for each round. Maybe after x rounds the point values double or triple. You decided how to code the scoring system, then implement your scoring system in your game.

## MATCH ME

### STUDENT DOCUMENT

```
1.1 1.2 *matchMe RAD X
*matchMe.py 81/82
corre
print

Previously coded

userC=[]
for i in range(10):
    if lst[i]==0:
        use
    elif lst[i]==1:
        use
    else:
        use
```

```
1.1 1.2 *matchMe RAD X
*matchMe.py 85/85
for i in range(10):
    if lst[i]==0:
        us
    elif lst[i]==1:
        us
    else:
        us

Previously Coded

print("Comp
print("User:"
```