

Unit Circle Cruncher

In this lesson, you will create two Unit Circle games using the Python Editor.

1) Your first game will randomly select a degree value from a list. It will randomly display either a cosine or sine question using that degree. The program will request the exact value on the unit circle for that cosine or sine value. If the answer entered is correct, game play will continue.

2) The second game is a modified version of the first game. Instead of randomly selecting a degree value from the list, the program will select a radian value from a list.

Objectives:

Programming Objectives:

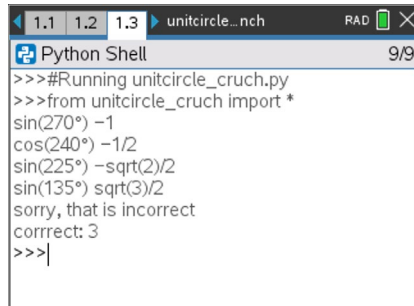
- Use the input function and a variable to collect and store data from a user
- Use the randint() function to generate random integers.
- Use a while loop to repeat code
- Use a list to store values

Math Objectives:

- Game 1: Evaluate sine and cosine values from the Unit Circle using degrees
- Game 2: Evaluate sine and cosine values from the Unit Circle using radians

In this project, you will create two Unit Circle Guessing games. The first version will request cosine and sine values in degree mode. The second version will request cosine and sine values in radian mode.

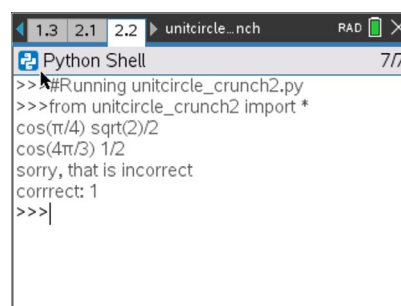
Version 1



```

Python Shell 9/9
>>>#Running unitcircle_crunch.py
>>>from unitcircle_crunch import *
sin(270°) -1
cos(240°) -1/2
sin(225°) -sqrt(2)/2
sin(135°) sqrt(3)/2
sorry, that is incorrect
correct: 3
>>>|
    
```

Version 2

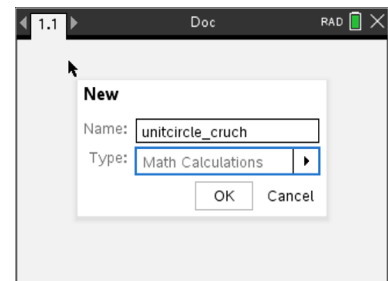


```

Python Shell 7/7
>>>#Running unitcircle_crunch2.py
>>>from unitcircle_crunch2 import *
cos(pi/4) sqrt(2)/2
cos(4pi/3) 1/2
sorry, that is incorrect
correct: 1
>>>|
    
```

1. Create a new python program named **unitcircle_crunch**.

Choose **Math Calculations** for the type.



2. You will need one more library, the random library.

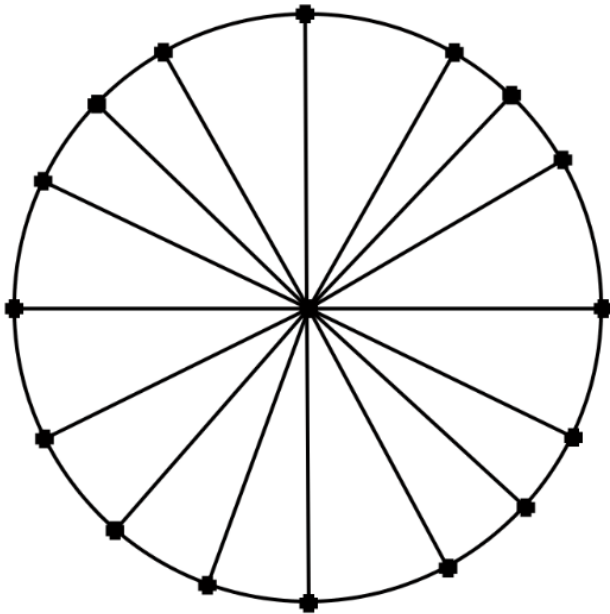
The random library contains the randint() function, which you will use to generate random integer coordinates.

Place your cursor on the line below the **import ti_plotlib**
Menu> Random> from random import *

```
1.1 | *Doc | RAD | X
unitcircle_cruch.py 5/6
# Math Calculations
=====
from math import *
from random import *
#=====
|
```

3. You will create a list of unit circle values.

Label all the degrees on the unit circle.



```
1.1 | *Doc | RAD | X
unitcircle_cruch.py 8/9
# Random Simulations
=====
from math import *
from random import *
#=====

degree=[0,30,45,
|
```

Add the values to a list named degrees.

degrees = [0, 30, 45, 60, #incomplete list add your values

4. The program will include a while loop that repeats while the questions are answered correctly. A variable named **flag** will store the value True or False to repeat or exit the loop.

The variable **correct** will keep track of the number of questions answered correctly.

```
1.1 | 1.2 | 1.3 | unitcircle_nch | RAD | X
unitcircle_cruch.py 1/33
# Random Simulations
=====
from math import *
from random import *
#=====

degree=[0,30,45,60,90,120,135,150,180,210,225
flag=True
correct=0
value=0
```

The variable **value** will store the value of the correct answer.

Add the lines:

```
flag = True
correct = 0
value = 0
```

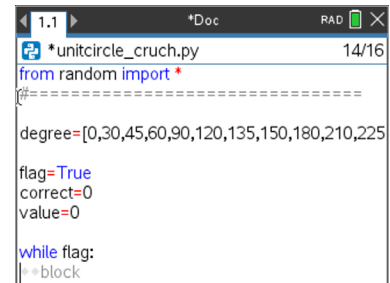
5. The program will repeatedly generate a new question, let the user enter an answer and check the validity. The while loop will allow this code to repeatedly execute while the value is true.

Menu> Built-ins> Control> while

while *BooleanExpr*:

◇◇block

The variable *flag* goes in the *BooleanExpr* placeholder. The rest of the code will go inside the block indentation



```
1.1 *Doc RAD X
*unitcircle_cruch.py 14/16
from random import *
#-----
degree=[0,30,45,60,90,120,135,150,180,210,225]
flag=True
correct=0
value=0
while flag:
|+=block
```

6. The items in the degree list are referenced using an index number. For example, `degree[0]` equals 0 because the number zero is in the first location of the list. The value of `degree[1]` is 30 because the number 30 is in the 1st index. Since the first index is at index 0, the last index of the list is one less than the length of the list.

To randomly select an item from the list you will:

*generate an index value from 0 to one less than the length of the list

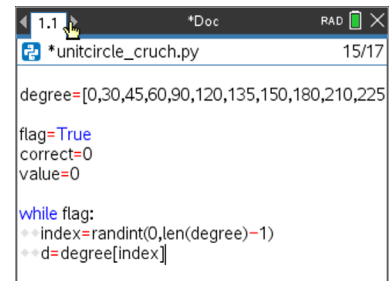
*use the index to select the degree from the list.

The `randint()` function can be accessed using

Menu> Random> randint

Add the lines:

```
index = randint( 0, len(degree)-1 )
d = degree[index]
```



```
1.1 *Doc RAD X
*unitcircle_cruch.py 15/17
degree=[0,30,45,60,90,120,135,150,180,210,225]
flag=True
correct=0
value=0
while flag:
  index=randint(0,len(degree)-1)
  d=degree[index]
```

7. Your program will either ask for a cosine value or a sine value.

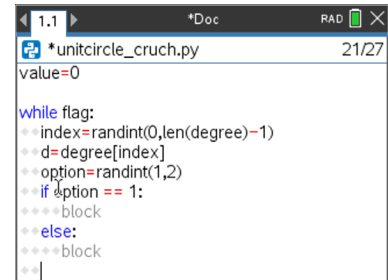
Use the randint() function to generate either a 1 or a 2. Store the value in a variable named **option**.

8. If the option is a 1, the code will display a cosine question. Otherwise, it will display a sine question.

Insert an if else statement.

Menu> Built-ins> Control> if else

In the if's BooleanExpr, check to see if the option is equal to 1.



```

1.1 | *Doc | RAD | X
*unitcircle_cruch.py | 21/27
value=0

while flag:
  index=randint(0,len(degree)-1)
  d=degree[index]
  option=randint(1,2)
  if option == 1:
    +++block
  else:
    +++block
  **

```

9. The program will ask a question such as: $\cos(30^\circ)$ and store the player's answer in the variable ans.

To ask the question, you join the string "cos(" with the variable d and the string "°)"

Because "cos(" is a string and d is a variable, you will combine the two by transforming d to string using str(d).

Menu> Built-ins> Type> Str()

The function eval() will be used to evaluate the value entered in the input() box.

Menu> Built-ins> I/O> eval

Menu> Built-ins> I/O> input

Add the line:

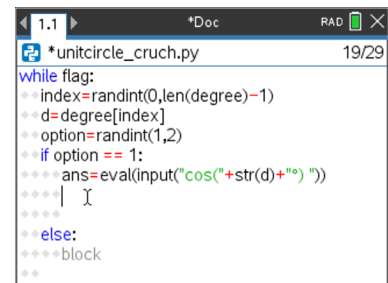
```
ans = eval(input("cos(" + str(d) + "°" ) ) )
```

10. To store the correct answer, add the line:

```
value=cos(radians(d))
```

This will store the evaluated cosine value in the variable value. You must put the radians() function around the d value to evaluate the function in degree mode.

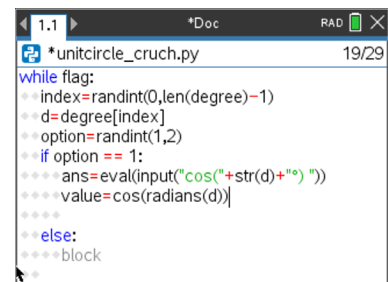
Menu> Math> Trig> radians()



```

1.1 | *Doc | RAD | X
*unitcircle_cruch.py | 19/29
while flag:
  index=randint(0,len(degree)-1)
  d=degree[index]
  option=randint(1,2)
  if option == 1:
    ans=eval(input("cos("+str(d)+"°" ))
    |  |
  else:
    +++block
  **

```



```

1.1 | *Doc | RAD | X
*unitcircle_cruch.py | 19/29
while flag:
  index=randint(0,len(degree)-1)
  d=degree[index]
  option=randint(1,2)
  if option == 1:
    ans=eval(input("cos("+str(d)+"°" ))
    value=cos(radians(d))
  else:
    +++block
  **

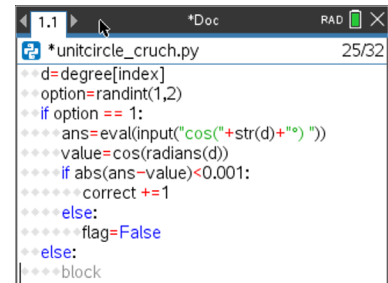
```

11. Now you need to compare the player's answer stored in **ans** with the actual answer stored in **value**. If the answers match, add 1 to the variable **correct**. If the answers don't match, change the flag to **False** so the loop will exit.

What do you think this if statement will look like?

12. You might have written:

```
if ans == value:
    correct += 1
else:
    flag = False
```



```
*unitcircle_cruch.py 25/32
d=degree[index]
option=randint(1,2)
if option == 1:
    ans=eval(input("cos("+str(d)+"°) "))
    value=cos(radians(d))
    if abs(ans-value)<0.001:
        correct +=1
    else:
        flag=False
else:
    block
```

While fundamentally correct, this will often cause a floating-point error to occur. For example, python will sometimes evaluate $\cos(60^\circ)$ to 0.499999999 or 0.500000001.

When this error occurs, the code will evaluate to **False** when it is actually true. To control for possible floating-point errors, write the if statement in the form:

```
if abs(ans - value) < 0.001:
    correct += 1
else:
    flag = False
```

13. Add the six lines of code for the sine function.

```
if option == 1:
    ans=eval(input("cos("+str(d)+"°) "))
    value=cos(radians(d))
    if abs(ans-value)<0.001:
        correct +=1
    else:
        flag=False
else:
```

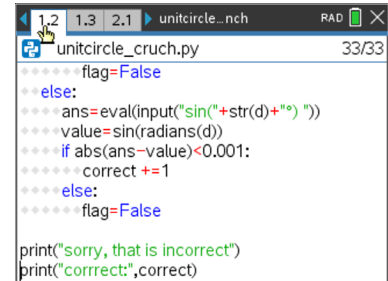
#add the lines of code for the sine function here

14. Once the user answer the question incorrectly, exit the loop.

Use code to print a message to let the user know the answer was incorrect.

Display the number of correct answers.

```
print("sorry, that is incorrect")
print("correct:",correct)
```



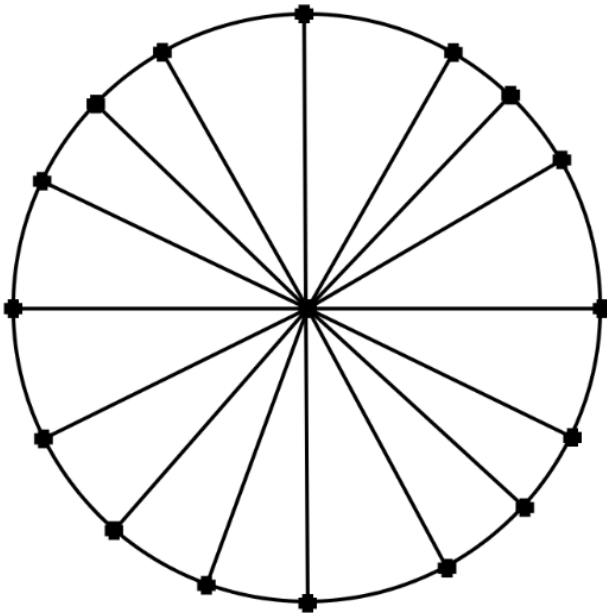
```
unitcircle_cruch.py 33/33
flag=False
else:
ans=eval(input("sin("+str(d)+"°)"))
value=sin(radians(d))
if abs(ans-value)<0.001:
correct +=1
else:
flag=False
print("sorry, that is incorrect")
print("correct:",correct)
```

15. *Play your game several times.*

Can you answer at least 10 questions in a row before the game terminates?

16. Let's make a second version to display the questions in radian mode.

Label all the radian measurements on the unit circle below.



17. Add a new Problem to the document.

Doc> Insert> Problem

Create a new python document named **unitcircle_crunch2**.

Leave the type blank.

Copy and Paste all the lines of code from the original program into this new document.

[ctrl] + [a] will select all lines

[ctrl] + [c] will copy the selected lines



[ctrl] + [v] will paste the lines

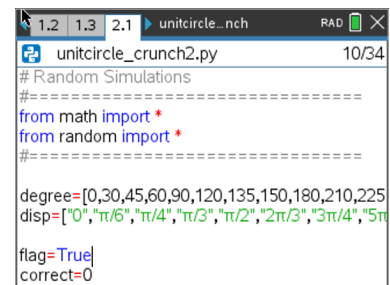
18. Below the line:

degree=[0,30,45,60,90,120,135,150,180,210.....

add your list of radian values

disp=["0","π/6","π/4","π/3","π/2","2π/3",...

Tech Tip You can type use the π button on the calculator, but it will write pi in it's place. If you want the π symbol, press the catalog button . The π symbol is under menu 4 .



```

unitcircle_crunch2.py 10/34
# Random Simulations
=====
from math import *
from random import *
=====
degree=[0,30,45,60,90,120,135,150,180,210,225]
disp=["0","π/6","π/4","π/3","π/2","2π/3","3π/4","5π/6"]
flag=True
correct=0
    
```

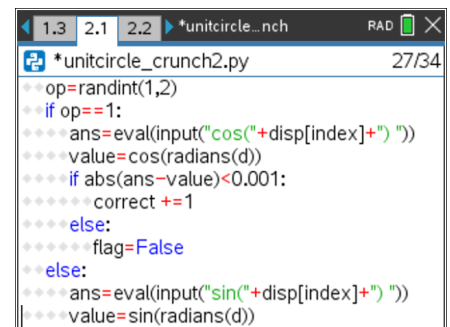
19. There are only two more lines that have to be modified. Which two lines do you think need modified?

20. The display line needs to display the corresponding radian instead of the degree. Change the ans = line to the following:

ans=eval(input("cos("+disp[index]+") "))

and

ans=eval(input("sin("+disp[index]+") "))



```

*unitcircle_crunch2.py 27/34
op=randint(1,2)
if op==1:
    ans=eval(input("cos("+disp[index]+") "))
    value=cos(radians(d))
    if abs(ans-value)<0.001:
        correct +=1
    else:
        flag=False
else:
    ans=eval(input("sin("+disp[index]+") "))
    value=sin(radians(d))
    
```

21. Can you get 10 in a row correct in radian mode?