WELCOME TO WEEK 4 OF INTRO TO PYTHON

QUICK REVIEW

- Types of errors & debugging them
- Conditional Statements:
  - *If Statements*
- Coding Blocks
- Random Numbers
- Relational Operators ”and” vs “or”

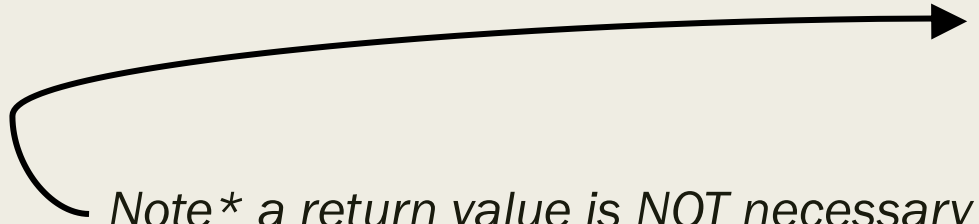
```
import random

def main():

    random.seed()

    # Roll both pairs of dice.
    score1 = rollPairDice()
    score2 = rollPairDice()

    print("Player 1 rolled a",score1,"and player 2 rolled a",score2)

    # Print out the winner.
    if score1 > score2:
        print("Player 1, you win!")
    else:
        print("Player 2, you win!")

def rollPairDice():
    return random.randint(1,6) + random.randint(1,6)

main()
```

CREATING FUNCTIONS

- Best practice is to create all code within functions and call when appropriate
  - *The base of your code is written in main()*
- Code that will be called repetitively or completes specified tasks can be defined in separate functions and called within *main()*

FUNCTION STRUCTURE

```
def myFunc(type variable):
    #your code here
    return someValue
```



Note a return value is NOT necessary*

REPETITIVE CONDITIONAL STATEMENTS

- Consists of *for* and *while* loops
- *while* loops work similar to *if* loops – checking if a statement is true in order to run
- *for* loops typically work with a counter and run up until a specified amount of times

LOOP SYNTAX

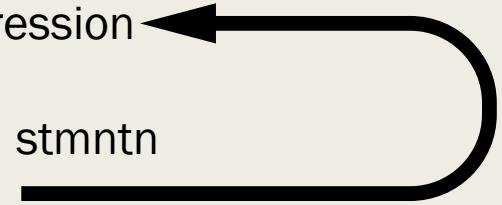
FOR LOOP

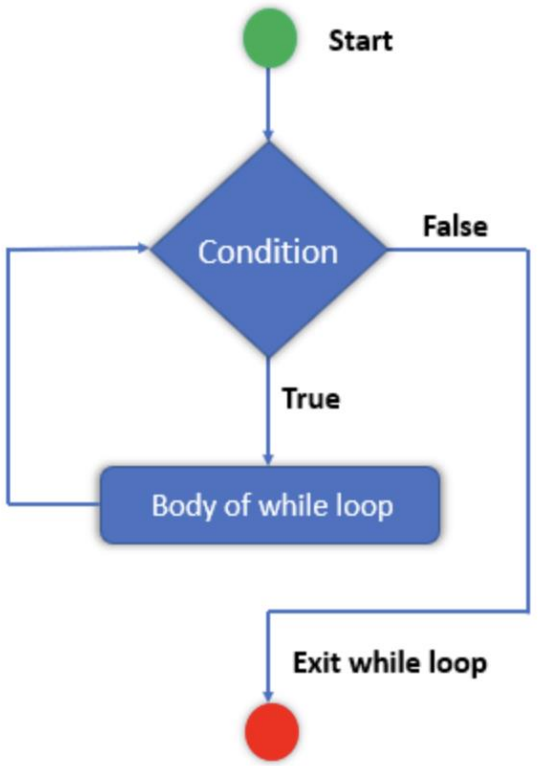
```
for <variable> in <range>:
    stmt1
    stmt2
    ...
    stmtn
```

WHILE LOOP

```
while <Boolean expression>:
    stmt1
    stmt2
    ...
    stmtn
stmtA
```

1. Evaluate Boolean expression
2. If true:
   a) Execute stmnt1 – stmntn
   b) Go back to step 1
3. If false, skip the over the loop and continue to stmtA




Start
Condition
False
True
Body of while loop
Exit while loop

WATCH OUT FOR THIS !!

Infinite loops are caused by not incrementing the counter value correctly within your segment of code.

Example:

```
def main():

    count = 1
    NUM_TIMES = 10

    while count <= NUM_TIMES:
        print(count,". I love programming in Python!", sep ="")
        count = count + 1

main()
```

NOTE* the variable we're checking against the Boolean statement doesn't have to be incrementing!

WHILE LOOP EXAMPLE

Write a program that generates a random number in between 1 and 100, inclusive, and then asks the user to make a guess. If their guess is too low, tell the user to guess higher. If it's too high, tell them to guess lower. When the user guesses the correct number, print out the number of guesses they took to do so and end the program.

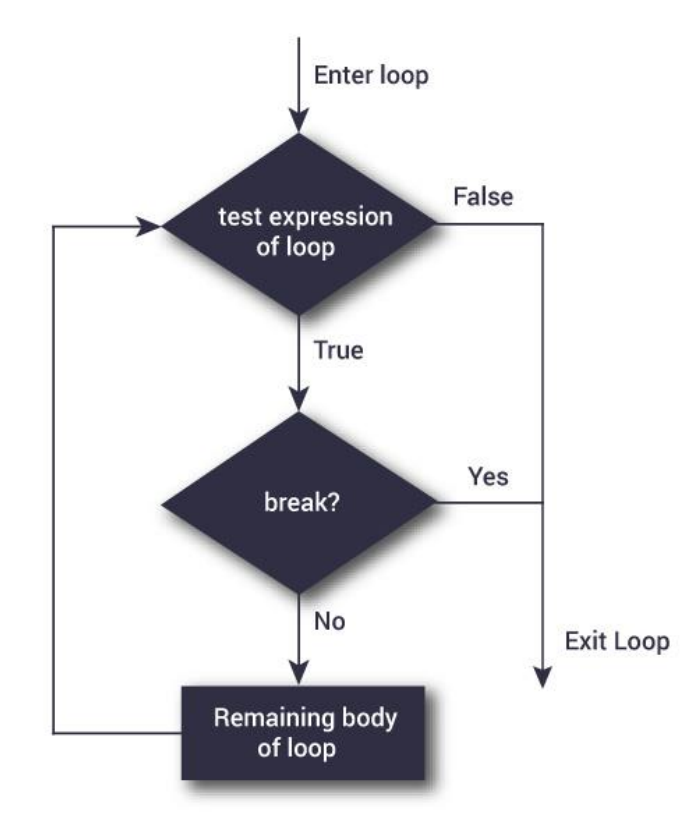
Sample Run

```
Enter your guess in between 1 and 100.
50
Your number is too high. Guess lower.
25
Your number is too low. Guess higher.
40
Your number is too low. Guess higher.
43
Great! You got the correct number 43 in 4 guesses!
```

LOOP CONTROL ELEMENTS

For loops that are tricky to design, We can better control their functionality using…

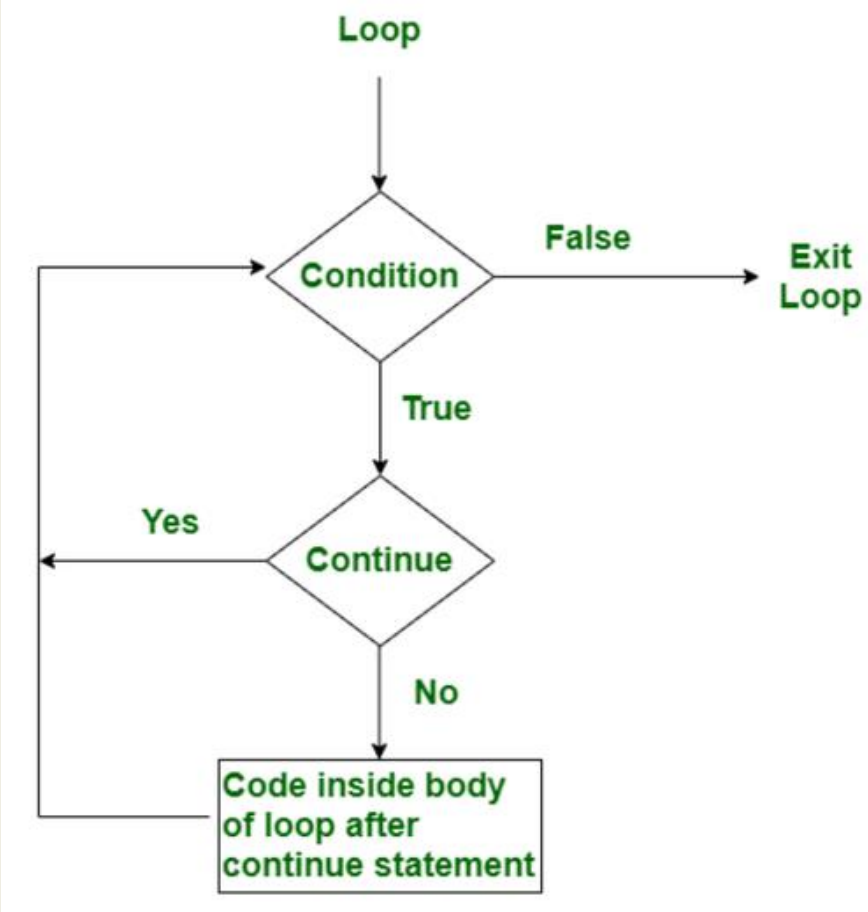
- *break*
  - Immediately ends a conditional loop
  - Typically used within an if statement
  - Can be used as an alternative to checking if a statement is true before starting a conditional statement
- *continue*
  - Skips over the rest of the current iteration of the loop and returns to the top of the loop
  - Typically used if an input can’t be processed and needs to be reentered
  - Must be used with either *break* or a counter to avoid an infinite loop




```
while True:

    stmt1
    ...
    if <exit condition1>:
        break
    stmt2
    ...
    if <exit condition2>:
        break
    stmt3
    ...
```

```
while <bool expr1>:
    stmt1
    if <bool expr2>:
        break
    stmt2

stmt3
```

BREAK STRUCTURE




```
while <bool expr1>:
    stmt1
    if <bool expr2>:
        continue
    stmt2

stmt3
```

CONTINUE STRUCTURE

INTRO TO TURTLE

- In the 1980s, a programming language called Logo was created to get kids interested in programming that allowed programmers to control a turtle that drew of the screen
- The turtle held a pen – that could be put down – and drew on the screen by moving forward and turning any number of degrees in either direction.
- Python implemented its own version of turtle to allow for visual displays
- More options are offered by Python's version:
  - *A variety of colors to draw with*
  - *Fill in shapes*
  - *So much more…* http://docs.python.org/library/turtle.html

GETTING STARTED WITH TURTLE

- To use turtle, you must include *import turtle* at the beginning of your program
- All functions available in turtle must be called using the syntax *turtle.function(<parameters>)*
- *Common Functions:*
  - *penup()* and *pendown()* : pick up and put down the turtle's pen
  - *forward(n)* : moves the turtle forward n pixles
  - *right(n)* and *left(n)* : alters the turtle's direction either right or left by n degrees

NOTE: forward, left, and right commands can be called regardless of the pen being put down or picked up !!

LET’S MERGE OUR KNOWLEDGE

Write a program that asks the user to enter the number of sides in a spiral triangle and then prints out a spiral triangle, very similar to the spiral square example.

SOME COOL TRICKS

- *fillcolor("color"), begin_fill()* and *end_fill()* : fill the desired portion of the graphic with a specified color
- *goto(x, y)* : move the turtle to position (x, y)
- *dot()* : leaves a dot in the current fill color
- To move backwards you can call the forward function with a negative number
  - *turtle.foward(-100)*

TEST YOUR CREATIVITY

Work in groups of two to program complete the turtle contest in JR Knights – you will work until 12:30pm in which time groups will present their work. The top 3 groups will get a prize !