CPSC 217,
Loops In Python: Part 2

- Branching vs repetition
- Nesting: branches with loops, loops with branches, loops within loops
- Introducing pseudo code
- The break instruction: how it works and why it should be used sparingly
- Logic errors that may occur with loops: endless loops
- Testing loops

James Tam

Common Mistake #1

- Mixing up branches (`IF` and variations) vs. loops (`while`)
- Related (both employ a Boolean expression) but they are not identical
- Branches
  - General principle: If the Boolean evaluates to true then execute a statement or statements (**once**)
  - Example: display a popup message if the number of typographical errors exceeds a cutoff.
- Loops
  - General principle: As long as (or while) the Boolean evaluates to true then execute a statement or statements (**multiple times**)
  - Example: While there are documents in a folder that the program hasn't printed then continue to open another document and print it.

James Tam

Common Mistake #1: Example

- **Program name**: `12branchVsLoop.py`
  - Learning objective: knowing the difference between a branching vs. an iterative (solution).

```
age = int(input("Age positive only: "))
if (age < 0):
    age = int(input("Age positive only: "))
print("Branch:", age)
```

Vs.

```
age = int(input("Age positive only: "))
while (age < 0):
    age = int(input("Age positive only: "))
print("Loop:", age)
```

James Tam

Recap: What You Know

- Branching: various forms (e.g. IF, IF-ELSE etc.) along with nested branches.
- Repetition: a single loop runs from start to end.

James Tam

Algorithm: Simple Loop, Repeat An Action

- This example (of something you know) will be used to help illustrate how the new concepts work.
- Pseudo code for shoveling the snow for a single residence (single loop)

```
While (sidewalk is not sufficiently shoveled)
    Shovel some snow
```

Optional link to a physical demonstration of the algorithm:
https://www.youtube.com/watch?v=-qDUiIzBuZk

James Tam

Nesting

- Recall: **Nested branches** (one inside the other)
  - Nested branches:

```
If (Boolean):
    If (Boolean):
        ...
```

- Branches and loops (`for`, `while`) can be nested within each other

```
# Scenario 1
loop (Boolean):
    if (Boolean):
        ...
```

```
# Scenario 2
    if (Boolean):
        loop (Boolean):
    ...
```

```
# Scenario 3
loop (Boolean):
    loop (Boolean):
        ...
```

James Tam

Scenario 1 Algorithm: A Choice (Branch) Each Time A Process Is Repeated (Loop)

- Pseudo code for shoveling the snow for a single residence (single loop)

```
While (sidewalk is not sufficiently shoveled)
   Shovel some snow
   if (very sweaty) then
      wipe brow
   endif
```

Optional link to a physical demonstration of the algorithm:
https://www.youtube.com/watch?v=FtGFszTjBJY

James Tam

Recognizing When Looping & Nesting Is Needed

- **Scenario 1**: As long some condition is met **a question will be asked** (branch = question).
  - Example: As the question is asked if the answer is invalid then an error message will be displayed.
    - **Example**: While the user entered an invalid value for age (too high or too low) then **if the age is too low** an error message will be displayed.
    - Type of nesting: an IF-branch nested inside of a loop

```
loop (Boolean):
    if (Boolean):
        ...
```

James Tam

IF Nested Inside A While

- **Program name**: 13nestingIFinsideWHILE.py
  - Learning objective: checking a condition during a repetitive process.

```
age = - 1
MIN_AGE = 1
MAX_AGE = 118
age = int(input("How old are you (1-118): "))
while ((age < MIN_AGE) or (age > MAX_AGE)):
    if (age < MIN_AGE):
        print("Age cannot be lower than", MIN_AGE, "years")
    #(Age for too high also possible (similar)
    age = int(input("How old are you (1-118): "))

print("Age=", age, "is age-okay")
```

James Tam

Scenario 2 Algorithm: When Condition Met (Branch) Repeat A Process (Loop)

- Pseudo code for a workday (vs. day off)

```
If (work day)
   while (work there is still work left)
      do some more work
Else
    do non-work stuff
endif
```

James Tam

Recognizing When Looping & Nesting Is Needed

- **Scenario 2**: If a question (Boolean expression for a branch) answers true then check if a process should be repeated.
  - **Example**: If the user specified the country of residence as Canada then **repeatedly prompt for the province of residence** as long as the province is not valid.
  - Type of nesting: a loop nested inside of an IF-branch

```
If(Boolean):
    loop():
        ...
```

James Tam

While Nested Inside An IF

- **Program name**: 14nestingWHILEinsideIF.py
  - A repetitive process that occurs given a condition has been met

```
country = ""
province = ""
VALID_PROVINCES = "BC, AB, SK, MB, ON, PQ,NL, NB, NS, PEI"
country = input("What is your country of citizenship: ")
if (country == "Canada"):
    province = input("What is your province of citizenship: ")
    while province not in (VALID_PROVINCES):
        print("Valid provinces: %s" %(VALID_PROVINCES))
        province = input("What is your province of citizenship: ")
    print("Country:", country, ", Province:",province)
```

James Tam

Scenario 3 Algorithm: Each Time A Repeated Process Begins (1st Outer Loop) Repeat 2nd Process (2nd Inner Loop)

- Pseudo code for shoveling the snow for a multiple residences (nested loop).

```
While (there are some residences to be shoveled)
   While (sidewalk is not sufficiently shoveled)
      Shovel some snow
      if (very sweaty) then
        wipe brow
      endif
```

Optional link to a physical demonstration of the algorithm:
https://www.youtube.com/watch?v=AwlWpSVv864

- **Important point with nested loops**:
  - For **each time that the outer loop runs** (e.g. go to a new location).
  - The **inner loop runs from start to finish** (e.g. start shoveling from start to finish).

James Tam

Recognizing When Looping & Nesting Is Needed

- **Scenario 3**: While one process is repeated**, repeat another process**.
  - More specifically: for each step in the first process **repeat the second process from start to end**
  - **Example:** While the user indicates that he/she wants to calculate another tax return prompt the user for income, **while the income is invalid repeatedly prompt for income**.
  - Type of nesting: a loop nested inside of an another loop

```
Loop():
    Loop():
        ...
```

James Tam

Nested Loop: Example Process In Pseudo Code

Each time we have a tax return to calculate

Do While (user wants to calculate another return)

  Do While (salary invalid)
    Get salary information
  End loop

For each client as long as salary invalid repeatedly prompt

Complete each of these steps from start to end

  Do While (investment income invalid)
    Get investment income
  End loop
End loop
  …

James Tam

While Nested Inside Another While

- **Program name**: 15nestingWHILEinsideWHILE.py
  - Learning objective: a repetitive process that repeats from start to end each time another repetitive process occurs.

```
MIN_INCOME = 0
runAgain = "yes"
while (runAgain == "yes"):
    print("CALCULATING A TAX RETURN")
    income = -1
    while (income < MIN_INCOME):
        income = int(input("Income $"))
    runAgain = input("To calculate another return enter 'yes': ")
```

James Tam

Practice Example #2: Nesting

1. Write a program that will count out all the numbers from one to six.
2. For each of the numbers in this sequence the program will determine if the current count (1 – 6) is odd or even.
   a) The program display the value of the current count as well an indication whether it is odd or even.

- Which Step (#1 or #2) should be completed first?

James Tam

Step #1 Completed: Now What?

- For each number in the sequence determine if it is odd or even.
- This can be done with the modulo (remainder) operator: %
  - An even number modulo 2 equals zero (2, 4, 6 etc. even divide into 2 and yield a remainder or modulo of zero).
  - `if (counter % 2 == 0): # Even`
  - An odd number modulo 2 does not equal zero (1, 3, 5, etc.)
- Pseudo code visualization of the problem

```
Loop to count from 1 to 6
    Determine if number is odd/even and display message
End Loop
```

  - Determining whether a number is odd/even is a part of counting through the sequence from 1 – 6, checking odd/even is nested within the loop

James Tam

The Break Instruction

- It is used to terminate the repetition of a loop which is separate from the main Boolean expression (it's another, separate Boolean expression).
- **General structure**:

```
for (Condition 1):
    if (Condition 2):
       break
```

```
while (Condition 1):
    if (Condition 2):
        break
```

James Tam

The Break Instruction (2)

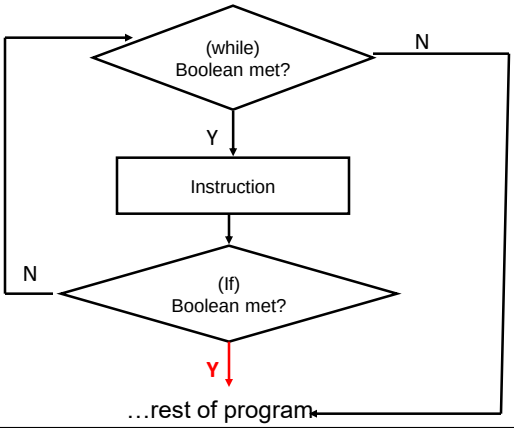
- **Program name**: 16break_illustration_only_avoid.py
  - Learning objective: early termination of a loop occurring any time in the loop body (most for illustration purposes).

```
MIN = 0
MAX = 9
number = random.randrange(MIN,MAX)+1
guess = -1
while (number != guess):
    print("Enter a number from %d-%d: " %(MIN+1,MAX+1), end="")
    guess = int(input())
    if (number == guess):
        print("Guessed correctly")
        break
    elif (guess < number):
        print("Higher.")
    else:
        print("Lower.")
print("Finished the game")
```

James Tam

The Break Should Be Rarely Used

- Adding an **extra exit point** in a loop (aside from the Boolean expression in the while loop) may make it harder to trace execution (leads to 'spaghetti' programming).



JT: While adding a single break may not always result in 'spaghetti' it's the beginning of a bad habit that may result in difficult to trace programs

James Tam

An Alternate To Using A 'Break'

- **NO:** Instead of an 'if' and 'break' inside the body of the loop

```
while (BE1):
    if (BE2):
       break
```

- **YES**: Add the second Boolean expression as part of the loop's main Boolean expression

```
while ((BE1) and not (BE2)):
```

James Tam

Another Alternative To Using A 'Break'

- **YES**: If the multiple Boolean expressions become too complex consider using a 'flag'

```
flag = True
while(flag == True):
    if(BE1):
        flag = False
    if(BE2):
        flag = False
    # Otherwise the flag remains set to true
    # BE = A Boolean expression
```

- Both of these approaches (YES #1 & 2)still provide the advantage of a single exit point from the loop.

James Tam

Alternative To Using Break

- **Third, complete and executable example**:
  17_break_alternative.py
  - A fully working example for you to look through on your own if you need to see a fully working alternative to using a break.
  - Snippet of the relevant part of the program:

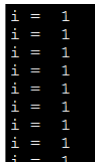
```
while(notDone == True):   #Alternative: while (notDone):
    print("Enter a number from %d-%d: " %(MIN+1,MAX+1),
      end="")
    guess = int(input())
    if(number == guess):
        print("Guessed correctly")
        notDone = False
```

James Tam

Infinite Loops

- Infinite loops never end (the stopping condition is never met).
- They can be caused by logical errors:
  - The loop control is never updated (Example 1 – below).
  - The updating of the loop control never brings it closer to the stopping condition (e.g. `i = i – 1 instead of  i = i + 1 in the example below`).
- **Program name**: `18infinite_never_updates.py`
- Learning objective: tracing a loop that never ends.

```
i = 1
while(i <= 10):
     print("i = ", i)
i = i + 1
```



To stop a program with an infinite loop in Unix simultaneously press the <ctrl> and the <c> keys

James Tam

Infinite Loops (2)

- **Program name**: `19infinite_update_no_closer_to_end.py`
- Learning objective: tracing a loop that never ends.

```
i = 10
while(i > 0):
    print("i = ",  i)
    i = i + 1
print("Done!")
```

James Tam

Testing Loops

- Make sure that the loop executes the proper number of times.
- Test conditions:
  1) Loop does not run
  2) Loop runs exactly once
  3) Loop runs exactly 'n' times

James Tam

Testing Loops: An Example

Program name: `20testing.py`

- Learning objective: minimum tests for a loop that steps through a sequence.

```
sum = 0
i = 1
last = 0

last = int(input("Enter the last number in the sequence to sum : "))
while(i <= last):
    sum = sum + i
    print("i = ", i)
    i = i + 1

print("sum =", sum)
```

James Tam

Reminder: Why Is Testing Important?

- Testing is done in actual practice.
- Determining “how did on an assignment”
- Because the marking key is posted ahead of time if you test your program thoroughly before submitting the final version then you should get a pretty clear idea of "how you will do".
  - Even if the marking for earlier assignments is not provided before the next assignment comes due you should already have a rough idea of your grade.

James Tam

Extra Practice #3

- Write a loop that will continue repeating if the user enters a value that is negative.
- Write a program that will prompt the user for number and an exponent. Using a loop the program will calculate the value of the number raised to the exponent.
  - To keep it simple you can limit the program to non-negative exponents.

James Tam

After This Section You Should Now Know

- How/when to employ nested branches and loops
  - How to trace their execution (branches with loops, loops with branches, loops within loops)
- The `break` instruction, why it should be avoided and alternatives to its use
- What is an infinite loop, some scenarios when they can occur.
- How to test loops (minimum test cases)

James Tam

Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."

James Tam