Loops In Python: Part 3

- Nesting: branches with loops, loops with branches, loops within loops
- The break instruction: how it works and why it should be used sparingly
- The continue instruction

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

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

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

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

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):
        ...
```

IF Nested Inside A While

- **Program name**: 1nestingIFinsideWHILE.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")
```

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
```

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():
        ...
```

While Nested Inside An IF

- **Program name**: 2nestingWHILEinsideIF.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)
```

Scenario 3 Algorithm: Each Time A Repeated Process Begins (1^{st} Outer Loop) Repeat 2nd Process (2^{nd} 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
```

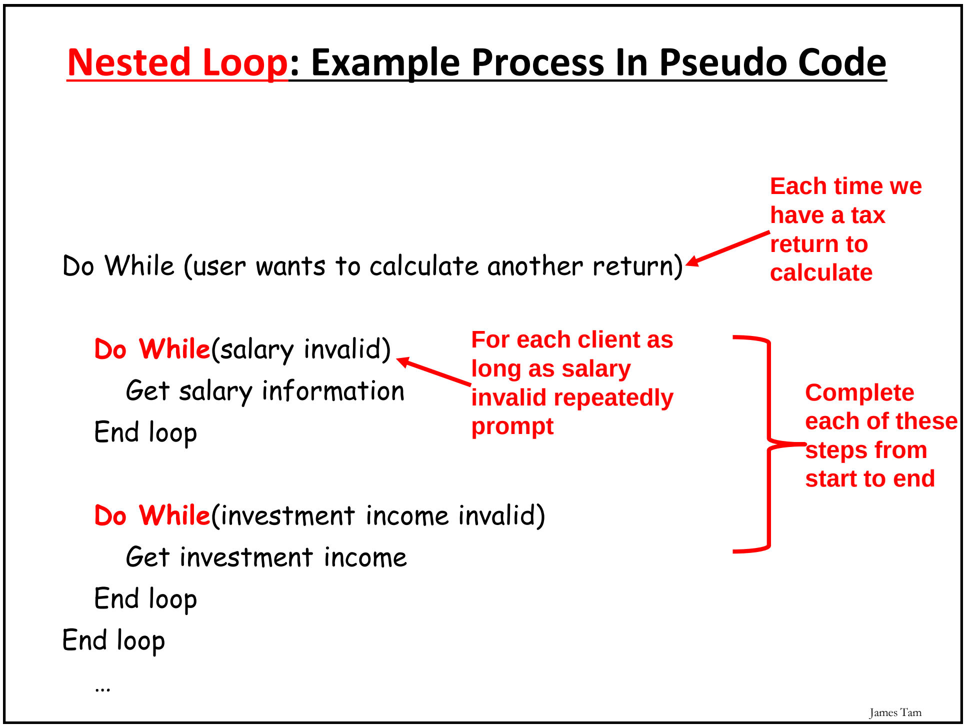
Optional link to a physical demonstration of the algorithm:
https://www.youtube.com/watch?v=AwlWpSVv864

- **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).

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():
        ...
```



Nested Loop: Example Process In Pseudo Code

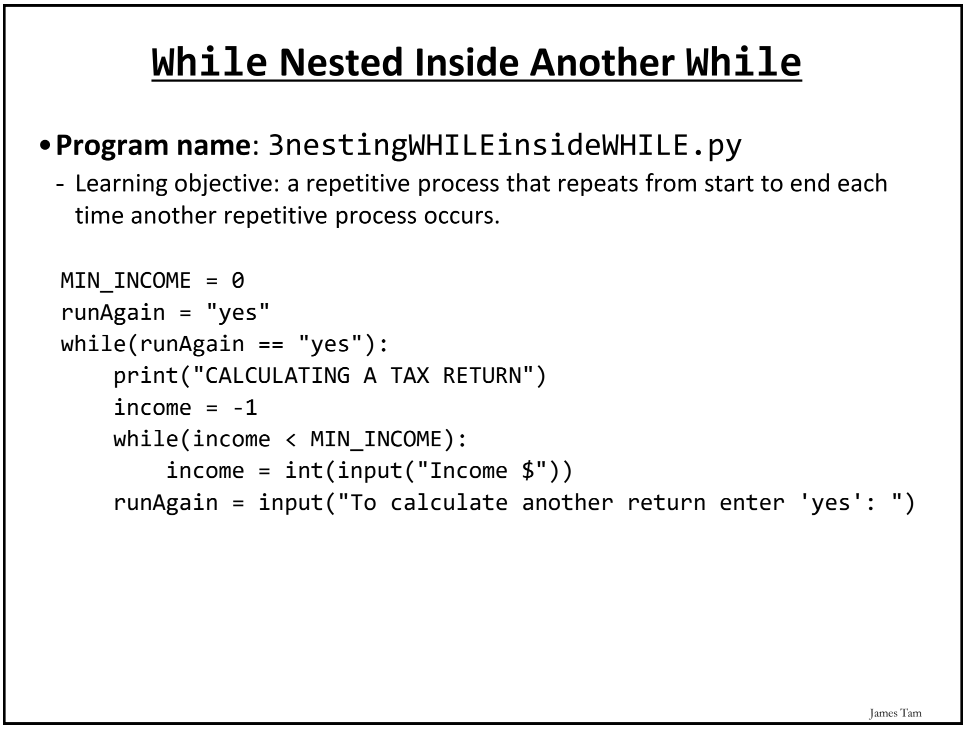
```
Do While (user wants to calculate another return)

    Do While(salary invalid)
        Get salary information
    End loop

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

Each time we have a tax return to calculate

For each client as long as salary invalid repeatedly prompt

Complete each of these steps from start to end

James Tam



While Nested Inside Another While

- **Program name**: `3nestingWHILEinsideWHILE.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

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

Using Other Python Libraries

- Python like other languages has a great deal of pre-written code.
- Some of it (such as `print()`, `input()`) are so common they are automatically imported with each program.
- Others must be manually imported
  - **Format:**

```
import <library/module name>
<library name>.<function or attribute name>
```

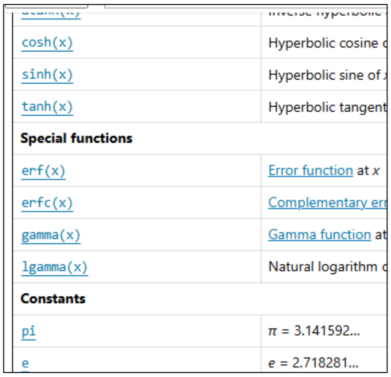
  - **Example:**

```
import math
print(math.pi)   #Access the constant attribute (JT: poor naming)
print(math.pow(2,3) #Calling pow function/method: 2 cubed
```

Some Python Libraries (‘Modules’)

Math constants & operations

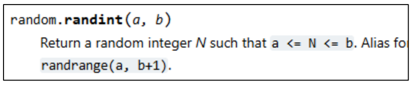
- Documentation: https://docs.python.org/3/library/math.html



atanh(x)	Inverse hyperbolic
cosh(x)	Hyperbolic cosine c
sinh(x)	Hyperbolic sine of x
tanh(x)	Hyperbolic tangent
Special functions	
erf(x)	Error function at x
erfc(x)	Complementary err
gamma(x)	Gamma function at
lgamma(x)	Natural logarithm c
Constants	
pi	π = 3.141592...
e	e = 2.718281...

Generating random numbers

- Documentation: https://docs.python.org/3/library/random.html
- (Included for reference as another library, not mandatory reading at this point as most of it is rather complex)



random.**randint**(*a*, *b*)

Return a random integer *N* such that `a <= N <= b`. Alias fo `randrange(a, b+1)`.

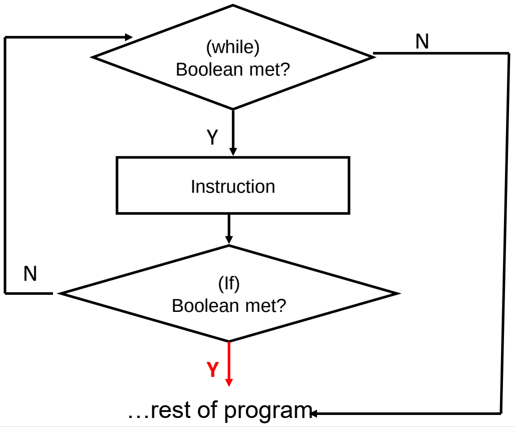
The Break Instruction (2)

- **Program name**: `4break_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 = 10
number = random.randint(MIN,MAX)
guess = -1
while(number != guess):
    print("Enter a number from %d-%d: " %(MIN+1,MAX), end="")
    guess = int(input())
    if(number == guess):
        print("Guessed correctly")
        break
    elif(guess < number):
        print("Higher.")
    else:
        print("Lower.")
print("Finished the game")
```

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

Another Reason For Avoiding Break

- (From past observations): when students were reminded that they were not to use a break they were baffled as to how to implement an alternative.
- In those cases the students could not write a moderately complex Boolean expression so they used a break to avoid working through that problem.
- Example algorithm (**DO NOT DO IT THIS WAY**)

```
Do while(Always true)
    if((BE1)and(BE2)) then
        break
    if(BE2):
        break
End while
```

  - (Before someone asks): working out the BE of the while without breaks would be a good practice exercise and you have an advantage that this is a practical and not a theory class: you can test sample solutions).
  - Hint: BE specifies the condition for the loop's execution whereas the breaks specify when the loop ends.

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

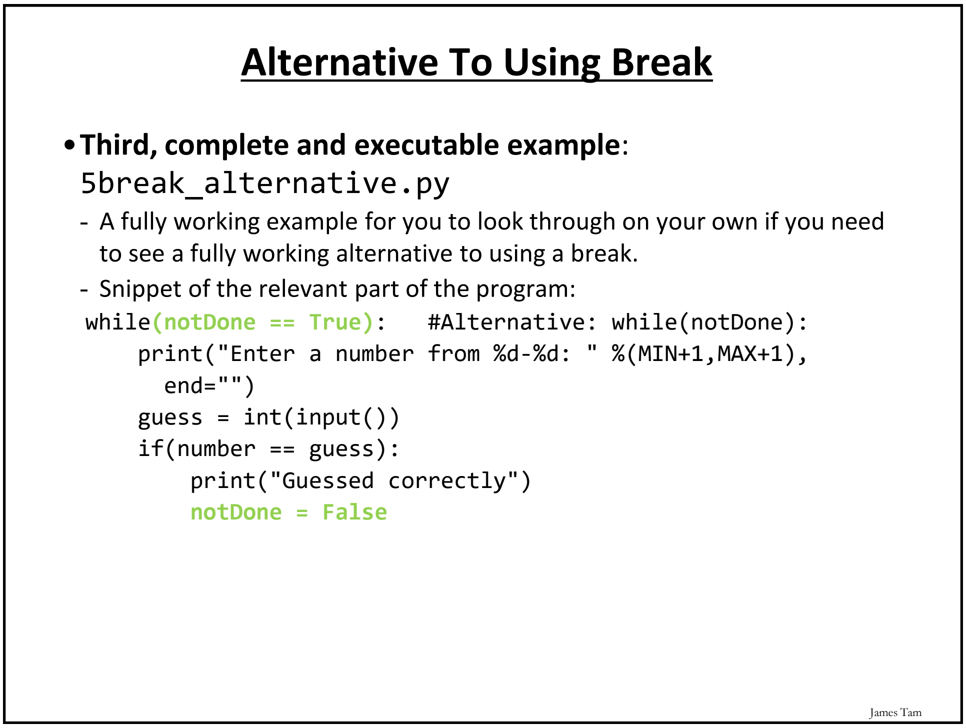
```
while((BE1) and not(BE2)):
```

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.

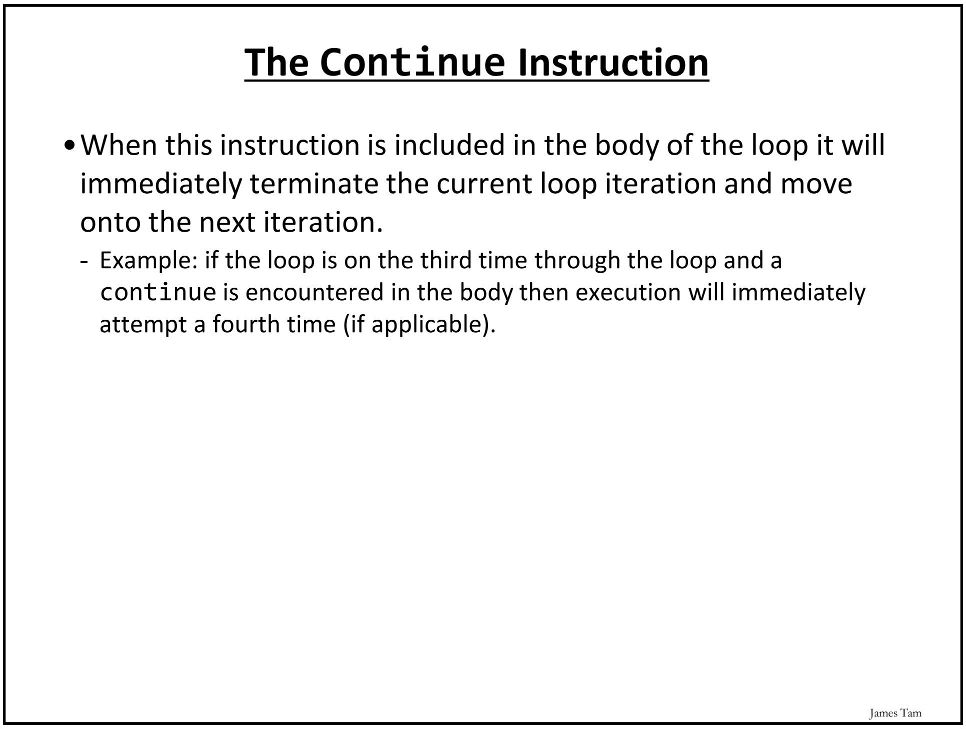


Alternative To Using Break

- **Third, complete and executable example**: `5break_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

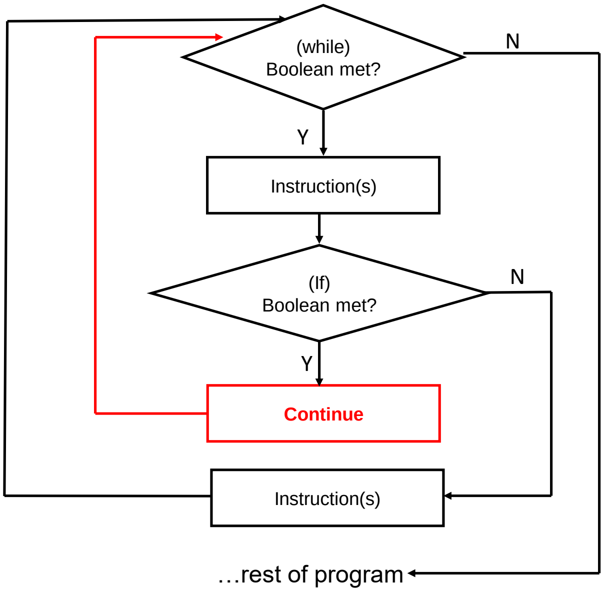


The Continue Instruction

- When this instruction is included in the body of the loop it will immediately terminate the current loop iteration and move onto the next iteration.
  - Example: if the loop is on the third time through the loop and a `continue` is encountered in the body then execution will immediately attempt a fourth time (if applicable).

James Tam

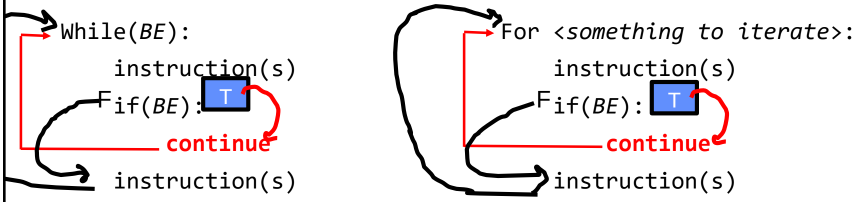
The Continue Instruction: Flowchart



The Continue Instruction: Pseudo-Code

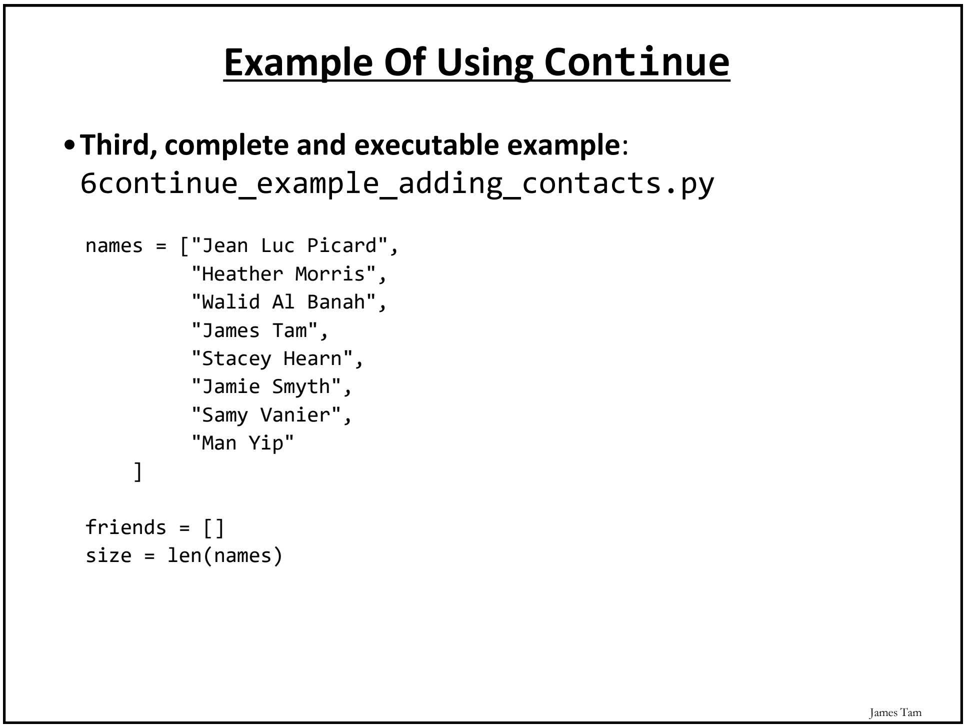
While

For



```
While(BE):
    instruction(s)
    if(BE):
        continue
    instruction(s)
```

```
For <something to iterate>:
    instruction(s)
    if(BE):
        continue
    instruction(s)
```

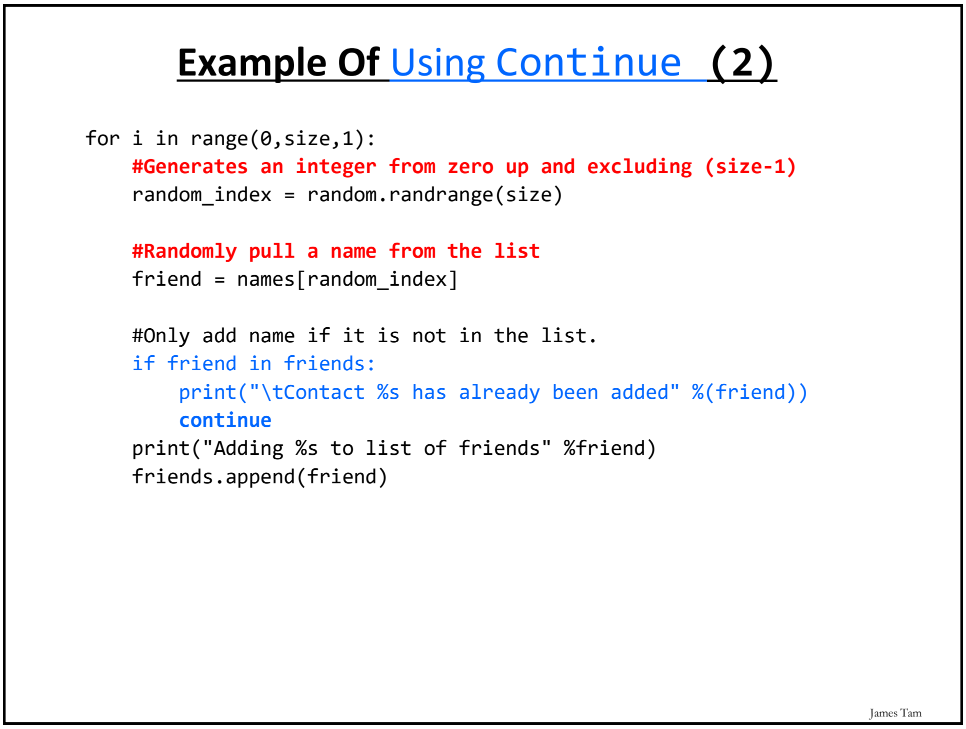


Example Of Using Continue

- **Third, complete and executable example**: `6continue_example_adding_contacts.py`

```
names = ["Jean Luc Picard",
         "Heather Morris",
         "Walid Al Banah",
         "James Tam",
         "Stacey Hearn",
         "Jamie Smyth",
         "Samy Vanier",
         "Man Yip"
    ]

friends = []
size = len(names)
```

James Tam



Example Of Using Continue (2)

```
for i in range(0,size,1):
    #Generates an integer from zero up and excluding (size-1)
    random_index = random.randrange(size)

    #Randomly pull a name from the list
    friend = names[random_index]

    #Only add name if it is not in the list.
    if friend in friends:
        print("\tContact %s has already been added" %(friend))
        continue
    print("Adding %s to list of friends" %friend)
    friends.append(friend)
```

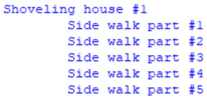
James Tam

Extra Example: Illustrating Nesting

- **Name of the full example:** `8nesting_shoveling_showing_example`
- **Learning:**
  - Tracing nested loops illustrated with nested loops.
  - 2 houses to shovel, each has 5 parts
  - Each time shoveling begins at a house, we have to start the process of shoveling part 1 – 5.
  - While implementation:

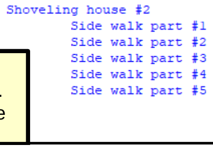
```
house = 1
while(house <=2):
    part = 1
    print(f"Shoveling house #{house}")
    while(part<=5):
        print(f"\tSide walk part #{part}")
        part = part + 1
    print()
    house = house + 1
```

1st time: outer loop



```
Shoveling house #1
        Side walk part #1
        Side walk part #2
        Side walk part #3
        Side walk part #4
        Side walk part #5
```

2nd time: outer loop



```
Shoveling house #2
        Side walk part #1
        Side walk part #2
        Side walk part #3
        Side walk part #4
        Side walk part #5
```

JT's hint learning how to trace nested loops

1) Trace only the **inner loop** in isolation (cut-paste the code if you have to).
2) Outer loop trace: recall the **outer body** runs from start-end each time the outer loop runs.

Extra Example: Illustrating Nesting (2)

- For implementation

```
for house in range(1,3,1):
    print(f"Shoveling house #{house}")
    for part in range(1,6,1):
        print(f"\tSide walk part #{part}")
    print()
```

Students-Do: Practice Exercise #1

- Write a loop that will repeatedly prompt if the user enters an age that is negative.
  - If an error condition occurs indicate to the user that the age cannot be less than zero.
- Write a second loop that will repeatedly prompt the for user's name if the nothing is empty i.e. the user just presses enter without entering a name.
  - Hint: here's one way of checking if the user enters a blank string

    ```
    aString = input()
    if(aString == ""):
        #Body
    ```
  - If an error condition occurs indicate to the user that the name cannot be blank.
- Only after a valid name and age have been entered display the following message:
  - `<`*`User enter age`*`> is a good age <`*`User entered name`*`>`
  - For instance if the user entered "smiley" for the name and "22" for the age then the program would display the following message>s
    - `22 is a good age smiley`

Students-Do: Practice Exercise #2

- Modify the previous program so after displaying a valid name and age it will prompt the user if they wish to enter another name and age.
  - As long as the user enters 'y' or 'Y' (i.e. case insensitive input) the program will repeat the algorithm of the previous program.

Students-Do: Practice Exercise #3

- Write a program that will using nested loops multiply all the products from 1x1 to 12x12 i.e. a “times table”.

James Tam

Students-Do: Practice Exercise #4

- Modify the following program so it draws a rectangle with the user specified number of rows and columns.

```
element = input("Type in the character used to draw the rectangle: ")
rows = int(input("Type in the number of rows: "))
columns = int(input("Type in the number of columns: "))
```

- **Solution to the exercise:** you can find it in the link on the course website with this week’s lecture materials.

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.
- How the `continue` instruction can be used.

Copyright Notification

- Unless otherwise indicated, all images in this presentation were provided courtesy of James Tam.