Introduction to Computer and Programming

Lecture 4

Yue Zhang

Westlake University

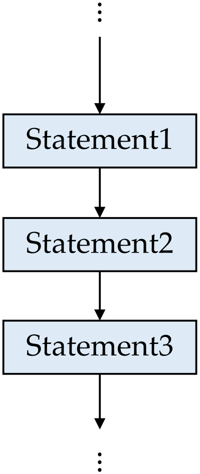
August 1, 2023

Chapter 4.

Branching and Looping

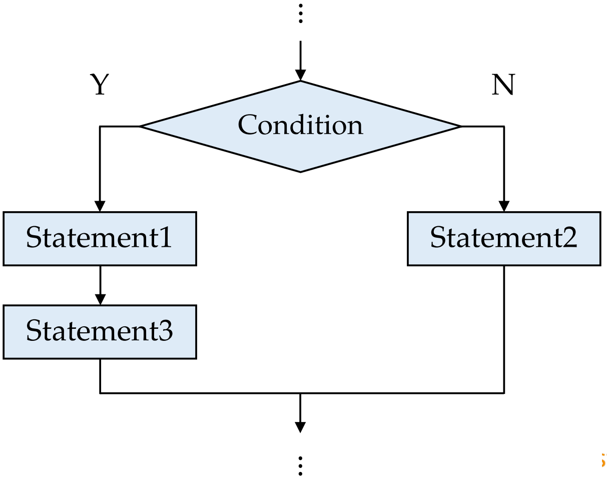
Order of Execution

- Sequential



Order of Execution

- Branching



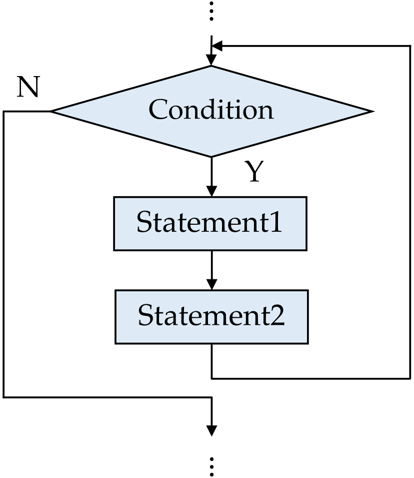
Order of Execution

Branching

- Different Behaviors
  according to different user's input or state
- e.g., checkbox, radio button
- game decisions

Order of Execution

- Looping



Order of Execution

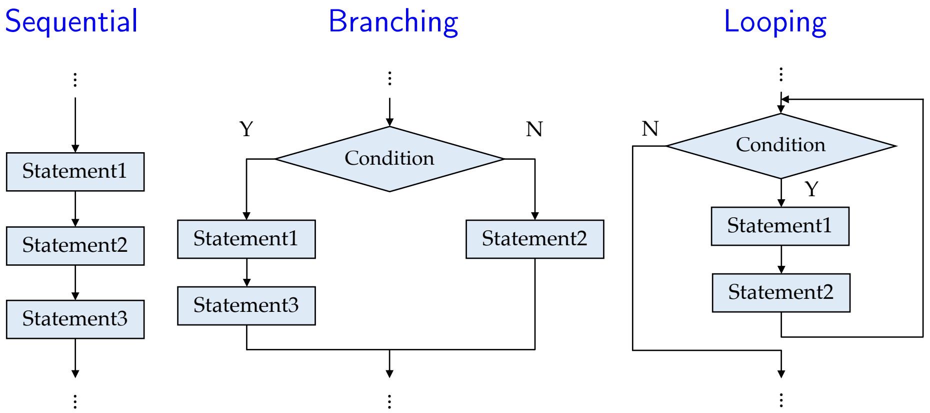
Looping

- Repeated Execution
- Multi-Step Task

  e.g., repeated play of music

    same code, repeated to play different music

Order of Execution



- Sufficient for achieving all functionalities by programs.

Order of Execution

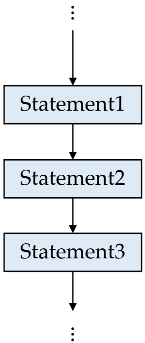
- What do programs do?
  - Input and output
  - Arithmatic and logical computation
  - Control flow

Order of Execution

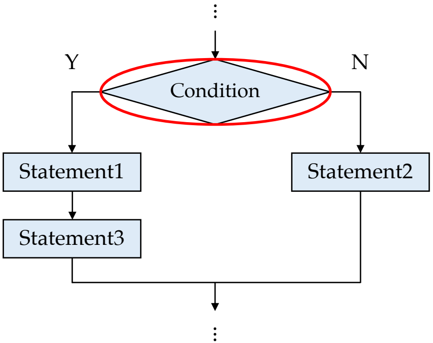
- What do programs do?
  - Input and output
  - Arithmatic and logical computation
  - Control flow
- That is basically all that a program does.

The Boolean Type

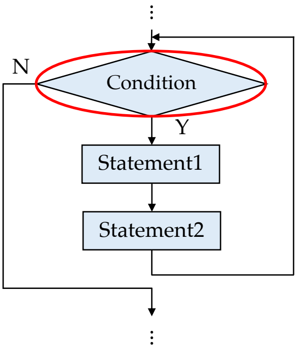
Sequential



Branching



Looping



The Boolean Type

- Literals

```
>>> a=True
>>> type(a)
<class 'bool'>
>>> b=False
>>> type(b)
<class 'bool'>
```

The Boolean Type

- Operators

```
>>> a=True
>>> b=False
>>> a and b
False
>>> a or b
True
>>> not a
False
>>> not b
True
```

WestlakeNLP

The Boolean Type

Truth Tables

and

x1	x2	x1 and x2
True	True	True
True	False	False
False	True	False
False	False	False

or

x1	x2	x1 or x2
True	True	True
True	False	True
False	True	True
False	False	False

not

x	not x
True	False
False	True

The Boolean Type

Truth Tables

and

x1	x2	x1 and x2
1	1	1
1	0	0
0	1	0
0	0	0

or

x1	x2	x1 or x2
1	1	1
1	0	1
0	1	1
0	0	0

not

x	not x
1	0
0	1

The Boolean Type

- Composite Expressions

```
>>> a=True
>>> b=True
>>> c=False
>>> a and not b
False
>>> c and b or a
True
>>> c and (b or a)
False
>>> a and b or b and c
True
```

Operator Precedence: not → and → or

The Boolean Type

- Operators Resulting in Boolean
  - numerical comparison

```
>>> 1==2
False
>>> 1!=2
True
>>> 3>5
False
>>> 3<=5
True
```

==, !=, >, <, >=, <=

The Boolean Type

- Infinity

```
>>> a=float('inf')
>>> b=float('-inf')
>>> a>1000000
True
>>> b<-10E9
True
```

float(“inf”), float(“-inf”)

The Boolean Type

- Operators Resulting in Boolean
  - strings and substrings

```
>>> s1='bc'
>>> s2='abcde'
>>> s3='b'
>>> s1 in s2
True
>>> s3 not in s2
False
>>> s1 in s3
False
>>> s1=='bc'
True
```

The *if* Statement

- Conditioned Execution

```
>>> if True:
...     print("Yes")
...
Yes
>>> if False:
...     print("Yes")
...
```

- if is a **compound** statement.

  if <Boolean Expression>:

  <statement block>

- a statement block has the same **indentations**.
  - a few spaces or tab keys.

The *if* Statement

- multiple statements in a statement block

```
>>> a=1
>>> b=2
>>> c=3
>>> if a>b:
...     d=a+b
...     print(d)
...
>>> if a>b or b<c:
...     d=b+c
...     print(d)
...
5
```

- all have the same indentation

  — a number of spaces or tabs

The *if* Statement

- Branching on User Input

zero.py

```python
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
print("Bye!")
```

The *if* Statement

- Branching on User Input

zero.py

```python
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
print("Bye!")
```

```
Yues~MacBook~Pro:code$ python zero.py
Give me a number: 0
The number is zero.
Bye!
Yues~MacBook~Pro:code$ python zero.py
Give me a number: 1
Bye!
```

The *if* Statement

- else for two-way branching

zero_else.py

```
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
else:
    print("The number is not zero.")
print("Bye!")
```

The *if* Statement

- else for two-way branching

zero_else.py

```python
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
else:
    print("The number is not zero.")
print("Bye!")
```

```
Yues~MacBook~Pro:code$ python zero_else.py
Give me a number: 1
The number is not zero.
Bye!
Yues~MacBook~Pro:code$ python zero_else.py
Give me a number: 0
The number is zero.
Bye!
```

The *if* Statement

- elif for multi-way branching

zero_elif.py

```
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
elif a > 0:
    print("The number is positive.")
else:
    print("The number is negative.")
print("Bye!")
```

WestlakeNLP

The *if* Statement

```
Yues~MacBook~Pro:code$ python zero_elif.py
Give me a number: 1
The number is positive.
Bye!
```

```
Yues~MacBook~Pro:code$ python zero_elif.py
Give me a number: -1
The number is negative.
Bye!
```

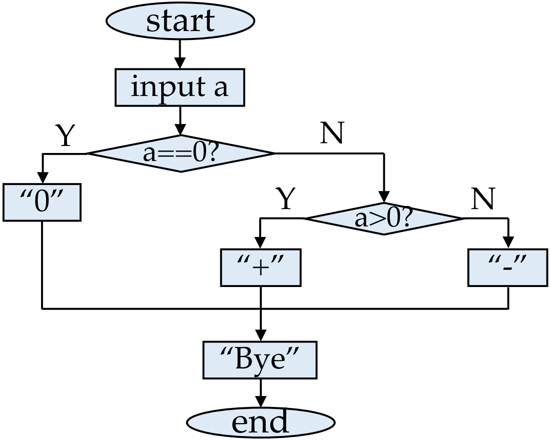
```
Yues~MacBook~Pro:code$ python zero_elif.py
Give me a number: 0
The number is zero.
Bye!
```

The *if* Statement

zero_elif.py

```python
a = int(input("Give me a number: "))
if a == 0:
    print("The number is zero.")
elif a > 0:
    print("The number is positive.")
else:
    print("The number is negative.")
print("Bye!")
```

Control Flow Diagram



The *if* Statement

- Nested if Statement

semi_final.py

```
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y":
    if win_2 == "Y":
        print("Gold Medal")
    else:
        print("Silver Medal")
else:
    if win_2 == "Y":
        print("Bronze Medal")
    else:
        print("Fourth Place.")
```

The *if* Statement

```
Yues~MacBook~Pro:code$ python semi_final.py
Did you win the first game? Y
Did you win the second game? Y
Gold Medal
```

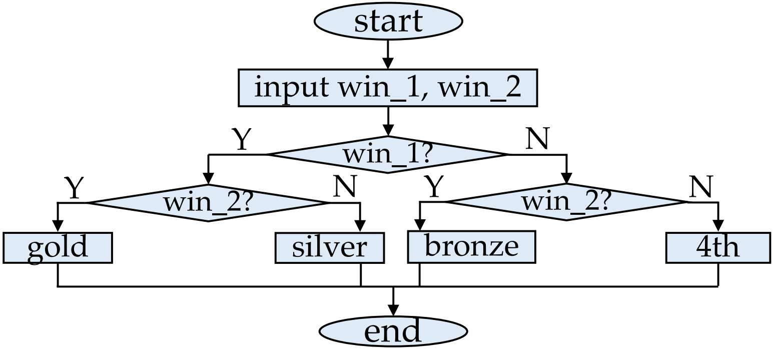
```
Yues~MacBook~Pro:code$ python semi_final.py
Did you win the first game? Y
Did you win the second game? N
Silver Medal
```

```
Yues~MacBook~Pro:code$ python semi_final.py
Did you win the first game? N
Did you win the second game? Y
Bronze Medal
```

```
Yues~MacBook~Pro:code$ python semi_final.py
Did you win the first game? N
Did you win the second game? N
Fourth Place.
```

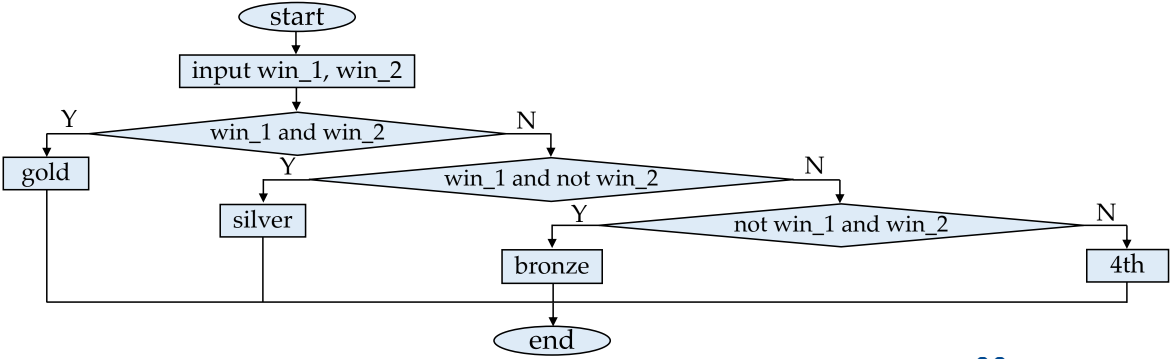
The *if* Statement

Control Flow Diagram



The *if* Statement

Equivalent Unnested Flow Diagram



The *if* Statement

- Unnested code

semi_final_unnested.py

```python
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

The *if* Statement

- Unnested code

semi_final_unnested.py

```
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

- Do you find flaws in this code?

The Ternary Operator

<value 1> if <boolean expression> else <value 2>

```
>>> a=1
>>> b=2
>>> c=1 if a>b else 0
>>> c
0
```

equivalent to:

```
>>> a=1
>>> b=2
>>> if a>b:
...     c=1
... else:
...     c=0
...
>>> c
0
```

The *while* Statement

```
>>> while False:
...     print("a")
...
>>> while True:
...     print("a")
a
a
a
a
^C
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
```

$\xrightarrow{\text{Note}}$ infinite loop = dead loop

$\xrightarrow{\text{keyboard}}$ Ctrl + C

while <Boolean Expression>
<statement block>

The *while* Statement

- Looping

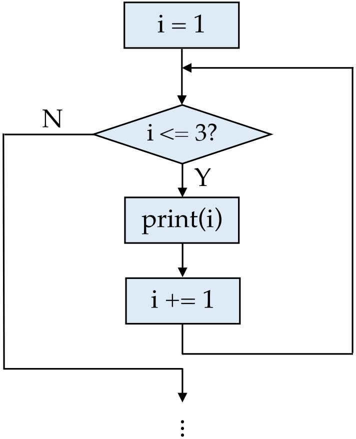
```
>>> i=1
>>> while i<=3:
...     print(i)
...     i+=1
...
1
2
3
```

The *while* Statement

- Looping

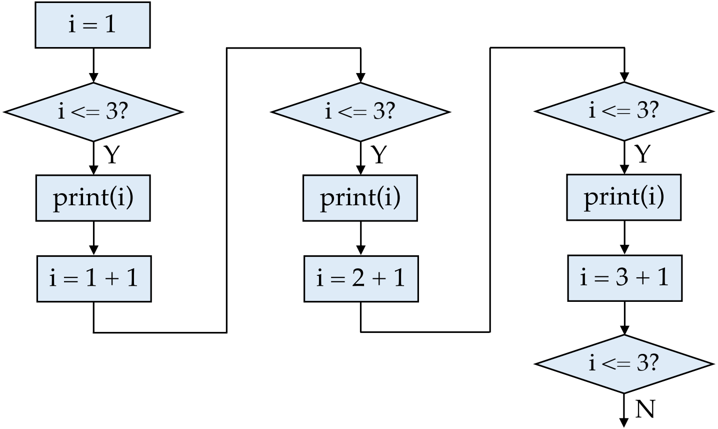
```
>>> i=1
>>> while i<=3:
...     print(i)
...     i+=1
...
1
2
3
```

Control Flow Diagram



The *while* Statement

Actual Execution *i* – loop variable



The *while* Statement

letter_gen.py

```python
import random   # the random module
s="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
command = input("Need a letter? (Y/N)\n")
while command == "Y":
    # random integer between 0 and length of s -1
    i = random.randint(0, len(s)-1)
    print('The letter is',s[i])
    command = input("Need a letter?  (Y/N)\n")
print("Bye")
```

The *while* Statement

```
Yues~MacBook~Pro:code$ python letter_gen.py
Need a letter? (Y/N)
Y
The letter is Y
Need a letter?  (Y/N)
Y
The letter is E
Need a letter?  (Y/N)
N
Bye
```

The *while* Statement

while and else

```
>>> while False:
...     print("a")
... else:
...     print("b")
...
b
```

else executed when <boolean expression> in while is False

The *while* Statement

while and else

```
>>> i=0
>>> while i<3:
...     print(i)
...     i+=1
... else:
...     print("Done")
...     print(i)
...
0
1
2
Done
3
```

Branching in a Loop

if in while

```
>>> i=1
>>> while i<100:
...     if i%7 == 0:
...         print(i)
...     i+=1
...
7
14
21
.
.
.
91
98
```

Branching in a Loop

guess.py

```python
import random
n = random.randint(0,100)  # the number to guess
wins = False
while not wins:
    guess = int(input("Your guess: "))
    if guess == n:
        print("You win!")
        wins = True
    elif guess > n:
        print("Too Large!")
    else:
        print("Too Small!")
```

Branching in a Loop

```
Yues~MacBook~Pro:code$ python guess.py
Your guess: 1
Too Small!
Your guess: 100
Too Large!
Your guess: 50
Too Large!
Your guess: 25
Too Small!
Your guess: 37
Too Small!
Your guess: 44
Too Large!
Your guess: 42
Too Small!
Your guess: 43
You win!
```

Branching in a Loop

guess_limit.py

```python
import random
limit = 5
guesses = 0
n = random.randint(0,100)  # the number to guess
wins = False
while not wins and guesses < limit:
    guess = int(input("Your guess: "))
    if guess == n:
        print("You win!")
        wins = True
    elif guess > n:
        print("Too Large!")
    else:
        print("Too Small!")
    guesses += 1
else:
    if not wins:
        print("You lose! The number is %d."%n)
```

Branching in a Loop

```
Yues~MacBook~Pro:code$ python guess_limit.py
Your guess: 26
Too Small!
Your guess: 40
Too Small!
Your guess: 80
Too Large!
Your guess: 60
Too Small!
Your guess: 70
Too Large!
You lose! The number is 64.
```

Break and Continue

- The else statement in while seems redundant.
  - The <Boolean Expression> will be False to stop while loop.
  - Thus you can just put what is in the else statement outside.

```
i = 1
while i <= 3:
    print(i)
    i += 1
else:
    print("done")
```

=

```
i = 1
while i <= 3:
    print(i)
    i += 1
print("done")
```

Break and Continue

- Jumping out of the while execution.

```
>>> i=1
>>> while i<=3:
...     print(i)
...     if i==2:
...         break   # jumps out
...     i+=1
... else:
...     print("I have counted to 3.")
...
1
2
>>> print(i)
2
```

Break and Continue

- Skipping the rest of one iteration.

```
>>> i=1
>>> while i<=3:
...     if i==2:
...         i+=1
...         continue    # jumps to while i <= 3
...     print(i)
...     i+=1
... else:
...     print("I have counted to 3.")
...
1
3
I have counted to 3.
>>> print(i)
4
```

Break and Continue

guess_limit.py

```
import random
limit = 5
guesses = 0
n = random.randint(0,100)  # the number to
     guess
wins = False
while not wins and guesses < limit:
    guess = int(input("Your guess: "))
    if guess == n:
        print("You win!")
        wins = True
    elif guess > n:
        print("Too Large!")
    else:
        print("Too Small!")
    guesses += 1
else:
    if not wins:
        print("You lose! The number is %d."%n)
```

guess_break.py

```
import random
limit = 5
guesses = 0
n = random.randint(0,100)  # the number to
     guess

while guesses < limit:
    guess = int(input("Your guess: "))
    if guess == n:
        print("You win!")
        break
    elif guess > n:
        print("Too Large!")
    else:
        print("Too Small!")
    guesses += 1
else:
    print("You lose! The number is %d."%n)
```

Break and Continue

```
Yues~MacBook~Pro:code$ python guess_break.py
Your guess: 20
Too Large!
Your guess: 10
Too Small!
Your guess: 15
Too Small!
Your guess: 17
Too Small!
Your guess: 19
You win!
```

Debugging

- Semi final code again

semi_final_unnested.py

```python
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

- Do you find flaws in this code?

Debugging

```
Yues~MacBook~Pro:code$ python semi_final_unnested.py
Did you win the first game? Yes
Did you win the second game? No
Fourth Place.
```

- expected – 'Silver Medal'

Debugging

- Tracing values

semi_final_trace.py

```
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
print("win_1 = " + win_1)           # trace value
print("win_2 = " + win_2)           # trace value
print("win_1 == 'Y' is " + str(win_1=='Y'))     # trace bool
print("win_1 == 'N' is " + str(win_1=='N'))     # trace bool
print("win_2 == 'Y' is " + str(win_2=='Y'))     # trace bool
print("win_2 == 'N' is " + str(win_2=='N'))     # trace bool

if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

Debugging

```
Yues~MacBook~Pro:code$ python semi_final_trace.py
Did you win the first game? Yes
Did you win the second game? No
win_1 = Yes
win_2 = No
win_1 == 'Y' is False
win_1 == 'N' is False
win_2 == 'Y' is False
win_2 == 'N' is False
Fourth Place.
```

Debugging

- Assertion

semi_final_assert.py

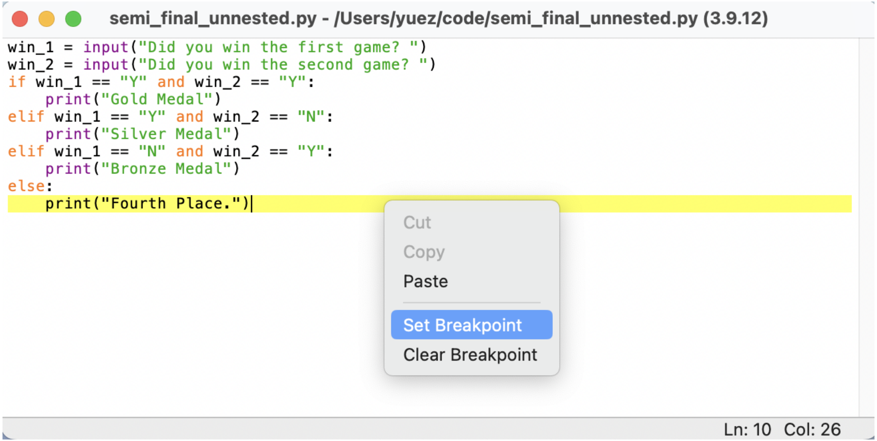
```python
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    # assert <Boolean Expression>
    assert win_1 == 'N' and win_2 == 'N'
    print("Fourth Place.")
```

Debugging

```
Yues~MacBook~Pro:code$ python semi_final_assert.py
Did you win the first game? Yes
Did you win the second game? No
Traceback (most recent call last):
  File "/Users/yuesz/code/semi_final_assert.py", line
      11, in <module>
    assert win_1 == 'N' and win_2 == 'N'
AssertionError
```

Debugging

- Debugging Tools – Break Points and Stepping

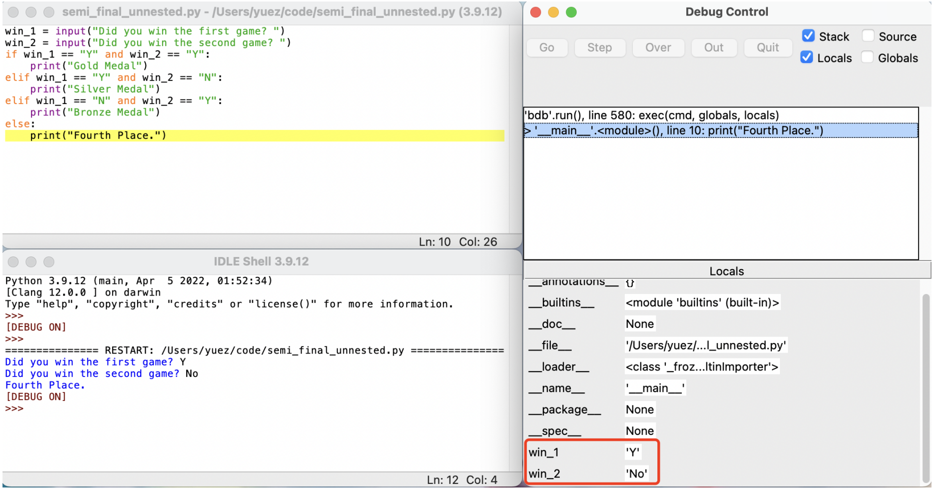


semi_final_unnested.py - /Users/yuez/code/semi_final_unnested.py (3.9.12)

```
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

Debugging

- Debugging Tools – Break Points and Stepping



semi_final_unnested.py - /Users/yuez/code/semi_final_unnested.py (3.9.12)

```python
win_1 = input("Did you win the first game? ")
win_2 = input("Did you win the second game? ")
if win_1 == "Y" and win_2 == "Y":
    print("Gold Medal")
elif win_1 == "Y" and win_2 == "N":
    print("Silver Medal")
elif win_1 == "N" and win_2 == "Y":
    print("Bronze Medal")
else:
    print("Fourth Place.")
```

Ln: 10 Col: 26

IDLE Shell 3.9.12

```
Python 3.9.12 (main, Apr  5 2022, 01:52:34)
[Clang 12.0.0 ] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
[DEBUG ON]
>>>
=============== RESTART: /Users/yuez/code/semi_final_unnested.py ===============
Did you win the first game? Y
Did you win the second game? No
Fourth Place.
[DEBUG ON]
>>>
```

Ln: 12 Col: 4

Debug Control

☑ Stack □ Source
☑ Locals □ Globals

```
'bdb'.run(), line 580: exec(cmd, globals, locals)
> '__main__'.<module>(), line 10: print("Fourth Place.")
```

Locals

__annotations__	{}
__builtins__	<module 'builtins' (built-in)>
__doc__	None
__file__	'/Users/yuez/...l_unnested.py'
__loader__	<class '_froz...ltinImporter'>
__name__	'__main__'
__package__	None
__spec__	None
win_1	'Y'
win_2	'No'

This week check-off:

Programming with branch and loop execution flow