Intro to IDLE, Variables, & Conditionals

Brandon Krakowsky

Penn Engineering

IDLE (Integrated Development and Learning Environment)

IDLE

- As you recall, Python comes bundled with IDLE (Python's Integrated Development and Learning Environment)
  - To download and install Python and IDLE, go here: https://www.python.org/downloads/ (Download the latest version)
- IDLE includes an interactive Python *interpreter* and *script editor*
  - We'll use the *script editor* to write and run Python *scripts*

Using IDLE – Keyboard Shortcuts

- To execute a command in the Python *shell*

  Press `Enter`
- To execute previous commands

  Use `CTRL + p` or `CTRL + n` to toggle commands in your history

  - To change this default setting, go to IDLE → Preferences → Keys
- To execute a script in the Python *script editor*

  Press `F5`

  On a Mac, press `FN + F5`
- To save a script in the Python *script editor*

  Press `CTRL + S`

  On a Mac, press `CMD + S`

Using IDLE - Shell

- You can type Python directly in the IDLE *shell*
- Print output to the console

```
print(“Hello World again!”)
print(‘Today’, ‘is another good day!’)
```

Using IDLE - Shell

- You can type Python directly in the IDLE *shell*
- Print output to the console

```
print(“Hello World again!”)
print(‘Today’, ‘is another good day!’)
```

- You can do math

```
2 + 3
2 * 3
2 * 3.5
```

Using IDLE - Shell

- You can type Python directly in the IDLE *shell*
- Print output to the console

```
print(“Hello World again!”)
print(‘Today’, ‘is another good day!’)
```

- You can do math

```
2 + 3
2 * 3
2 * 3.5
```

- True or False?

```
1 == 2
1 < 2
1.2 >= 1.2
```

Using IDLE – Running a Python Script

- To create and run a Python script in IDLE, you must first create and save a script file
  - Go to “File” --> “New File”
  - Go to “File” --> “Save”
- To open an existing script
  - Go to “File” --> “Open…”
- To execute a script
  - Go to “Run” --> “Run Module”

Python Scripts

What is a Python Script?

- A Python *script* is multiple lines of code stored in a file, also called a *module*
  - Modules help keep your code organized!
  - You can have similar or related code all in a single file

What is a Python Script?

- A Python *script* is multiple lines of code stored in a file, also called a *module*
  - Modules help keep your code organized!
  - You can have similar or related code all in a single file
- Python *scripts* have conventions
  - Some allow your code to execute correctly
  - Others help to keep your code readable

What is a Python Script?

- A Python *script* is multiple lines of code stored in a file, also called a *module*
  - Modules help keep your code organized!
  - You can have similar or related code all in a single file
- Python *scripts* have conventions
  - Some allow your code to execute correctly
  - Others help to keep your code readable
- Basic conventions
  - Put every statement on a line by itself
  - Use spaces around operators
  - Use spaces after commas, colons, and semicolons

What is a Python Script?

- A Python *script* is multiple lines of code stored in a file, also called a *module*
  - Modules help keep your code organized!
  - You can have similar or related code all in a single file
- Python *scripts* have conventions
  - Some allow your code to execute correctly
  - Others help to keep your code readable
- Basic conventions
  - Put every statement on a line by itself
  - Use spaces around operators
  - Use spaces after commas, colons, and semicolons
- As you learn more commands, you will learn more conventions!

Adding Comments to Python Scripts

- Use # to comment a single line of text

```
#Here's a single line comment
```

Adding Comments to Python Scripts

- Use # to comment a single line of text

```
#Here's a single line comment
```

- Use """…""" (3 double quotation marks) or use '''…''' (3 single quotation marks) to comment multiple lines at once

```
"""Here's a
   multi-line comment
   on multiple lines"""
```

Adding Comments to Python Scripts

- Use # to comment a single line of text

```
#Here's a single line comment
```

- Use """…""" (3 double quotation marks) or use '''…''' (3 single quotation marks) to comment multiple lines at once

```
"""Here's a
   multi-line comment
   on multiple lines"""
```

- Adding comments to a script keeps your code organized & readable!
  - Comments can be used to document your code

Adding Comments to Python Scripts

- Use # to comment a single line of text

```
#Here's a single line comment
```

- Use """…""" (3 double quotation marks) or use '''…''' (3 single quotation marks) to comment multiple lines at once

```
"""Here's a
   multi-line comment
   on multiple lines"""
```

- Adding comments to a script keeps your code organized & readable!
  - Comments can be used to document your code
- At the very least, you should add comments to all non-trivial lines of code

Adding Comments to Python Scripts - Example

- Here's an example program to greet a user

```
"""This code will:
Prompt for your name
Set the 'name' variable
Print a message"""
name = input("What is your name? ") #gets user's name
print("Hello {}".format(name)) #prints message with user's name
```

Note: To execute a script, Press F5 (on Mac, press FN + F5)

Variables

Variables

- A *symbolic* name for (or reference to) a value
  - When a variable gets assigned a value, it takes the *type* of the value
  - You can use variables to store all kinds of stuff!
  - Variable names are case sensitive

```
x = 2
y = 3
print(x)
print(y)
```

Note: Commands in light blue can be typed directly into the IDLE shell or a script file

Variables

- Python is *dynamically* typed
  - If you assign the same variable a different value later on in your code, the variable is updated with the new value

```
x = y
y = x + 3
x += 1 #increment x by 1, same as x = x + 1
y -= 1 #decrement y by 1, same as y = y - 1
print(x)
print(y)
```

Note: Commands in light blue can be typed directly into the IDLE shell or a script file

Variables

- Python is *dynamically* typed
  - If you assign the same variable a different value later on in your code, the variable is updated with the new value

```
x = y
y = x + 3
x += 1 #increment x by 1, same as x = x + 1
y -= 1 #decrement y by 1, same as y = y - 1
print(x)
print(y)
```

- Variables can change types

```
x = 5
x = ‘five’ #x went from holding an int to holding a string
print(type(x))
```

Note: Commands in light blue can be typed directly into the IDLE shell or a script file

Variables – Boolean Operators

- Using and operator

```
x = 5
print(x > 3 and x < 5)
```

Variables – Boolean Operators

- Using and operator

```
x = 5
print(x > 3 and x < 5)
```

- The same as this

```
print(3 < x < 5)
```

Variables – Boolean Operators

- Using and operator

```
x = 5
print(x > 3 and x < 5)
```

- The same as this

```
print(3 < x < 5)
```

- Using or operator

```
y = 3
print(y < 3 or y == 3)
```

Variables – Boolean Operators

- Using and operator
```
x = 5
print(x > 3 and x < 5)
```
- The same as this
```
print(3 < x < 5)
```
- Using or operator
```
y = 3
print(y < 3 or y == 3)
```
- Same as this
```
print(y <= 3)
```

Variables – Boolean Operators

- Using and operator
```
x = 5
print(x > 3 and x < 5)
```
- The same as this
```
print(3 < x < 5)
```
- Using or operator
```
y = 3
print(y < 3 or y == 3)
```
- Same as this
```
print(y <= 3)
```
- Using not operators
```
res = (y <= x)
print(not res) #prints the opposite of res, i.e. changes True to False
or False to True
```

Variables – Fancy Variable Assignment

- Assigns the same value to multiple variables

```
x = y = z = 5 #x and y and z are all set to 5
print(x, y, z)
```

Variables – Fancy Variable Assignment

- Assigns the same value to multiple variables

```
x = y = z = 5 #x and y and z are all set to 5
print(x, y, z)
```

- Assigns multiple values to multiple variables

```
x, y = ‘abc’, True #x is set to ‘abc’ and y is set to True
print(x, y)
```

Variables – Fancy Variable Assignment

- Assigns the same value to multiple variables

```
x = y = z = 5 #x and y and z are all set to 5
print(x, y, z)
```

- Assigns multiple values to multiple variables

```
x, y = ‘abc’, True #x is set to ‘abc’ and y is set to True
print(x, y)
```

- Swaps variable values

```
x, y = y, x #x is set to value of y and y is set to value of x
print(x, y)
```

Variables – Name Swap Exercise

- Create variables with your first, middle, and last name

```
first_name = 'Brandon’
middle_name = 'Lee’
last_name = "Krakowsky”
```

Variables – Name Swap Exercise

- Create variables with your first, middle, and last name

```
first_name = 'Brandon’
middle_name = 'Lee’
last_name = "Krakowsky”
```

- Print your full name

```
print(first_name, middle_name, last_name)
```

Variables – Name Swap Exercise

- Create variables with your first, middle, and last name
```
first_name = 'Brandon’
middle_name = 'Lee’
last_name = "Krakowsky”
```
- Print your full name
```
print(first_name, middle_name, last_name)
```
- Swap your first and last name
```
first_name, middle_name, last_name = last_name, middle_name, first_name
```

Variables – Name Swap Exercise

- Create variables with your first, middle, and last name
```
first_name = 'Brandon’
middle_name = 'Lee’
last_name = "Krakowsky”
```
- Print your full name
```
print(first_name, middle_name, last_name)
```
- Swap your first and last name
```
first_name, middle_name, last_name = last_name, middle_name, first_name
```
- Print your full name again
```
print(first_name, middle_name, last_name)
```

Variables – Variable Substitution

- In mathematical expressions

```
x = y = 10
z = 2 * x + y
print(z)

z = z ** (y - 1)
print(z)
```

Variables – Variable Substitution

- In mathematical expressions

```
x = y = 10
z = 2 * x + y
print(z)

z = z ** (y - 1)
print(z)
```

- In boolean expressions

```
x = 42
b = 15 < (x / 2) < 25
print(b) #what is b?
print(type(b)) #what type is b?
```

Variables – Variable Substitution

- In multiplication

```
x = 42
y = str(x)
x *= 2 #same as x = x * 2
y *= 2 #same as y = y * 2
print(x) #what does this display?
print(y) #and this?
```

Combining Variables

- Store string values

```
fav_movie = “Justin Bieber’s Believe”
fav_singer = “Justin Bieber”
```

Combining Variables

- Store string values

```
fav_movie = “Justin Bieber’s Believe”
fav_singer = “Justin Bieber”
```

- Then combine them to create a new string variable

```
favs = “Your favorite movie is “ + fav_movie + “ and your favorite singer is “ + fav_singer
```

- Print the favs variable

```
print(favs)
```

Variables – VERY Simple Exercise

- Set a variable x to “cats”
- Set a variable y to “dogs”
- Referencing the variables above, set a new variable s to “It’s raining cats and dogs!”
- Print s

Variables – VERY Simple Exercise

- Set a variable x to “cats”
  ```
  x = “cats”
  ```
- Set a variable y to “dogs”
  ```
  y = “dogs”
  ```
- Referencing the variables above, set a new variable s to “It’s raining cats and dogs!”
  ```
  s = “It’s raining “ + x + “ and “ + y + “!”
  ```
- Print s
  ```
  print(s)
  ```

Variables – Getting User Input

- Use *input* to prompt the user

```
input(“What is your favorite movie?”)
```

Variables – Getting User Input

- Use *input* to prompt the user

```
input(“What is your favorite movie?”)
```

- You can dynamically set a variable based on user input

```
fav_movie = input(“What is your favorite movie?”)
fav_singer = input(“Who is your favorite singer?”)
```

Variables – Getting User Input

- Use *input* to prompt the user

```
input(“What is your favorite movie?”)
```

- You can dynamically set a variable based on user input

```
fav_movie = input(“What is your favorite movie?”)
fav_singer = input(“Who is your favorite singer?”)
```

- And combine them to create a new string variable using “string interpolation” using the *format* function

```
favs = “Your favorite movie is {} and your favorite singer is
{}”.format(fav_movie, fav_singer)
```

  - The curly braces {} indicate placeholders for the fav_movie and fav_singer values

- Print the favs variable again

```
print(favs)
```

Variables – Getting User Input

- The *input* command returns a string by default

```
age = input(“How old are you?”)
print(type(age))
```

Variables – Getting User Input

- The *input* command returns a string by default

```
age = input(“How old are you?”)
print(type(age))
```

- Now calculate how old you’ll be in 3 years

```
print(age + 3)
```

Variables – Getting User Input

- The *input* command returns a string by default

```
age = input("How old are you?")
print(type(age))
```

- Now calculate how old you'll be in 3 years

```
print(age + 3)
```

- Whoops, you should have received an error!
  - age is a string and you're trying to add it to an int
  - Cast it first!

```
age = int(input("How old are you?"))
print(age + 3)
```

Variables - Exercise

- Write code to calculate the total bill at a restaurant (meal + tax + tip)
  - Prompt the user for the *bill amount*, the *sales tax*, and the *tip percentage*
  - Print out the total bill amount in the format: "The total bill is $..."
- Make sure you:
  - Apply the tip after you add the tax amount
  - Round the total bill amount to 2 decimal places
- For example, let's say you went to a nice restaurant and ate a $30 meal. PA's sales tax is 6%, and you want to tip the waiter 18%.
  - This prints: "The total bill is $37.52"

Variables - Exercise

- Write code to calculate the total bill at a restaurant (meal + tax + tip)

```
#define variables from user input
bill = float(input('How much is the meal? '))
tax = float(input('What is the sales tax (percentage)? '))
tip = float(input('How much of a tip (percentage)? '))

tax_amount = (bill * tax) / 100 #calculate tax
total = bill + tax_amount #calculate bill amount, without tip

tip_amount = (total * tip) / 100 #calculate tip amount
total = total + tip_amount #calculate total bill amount

total = round(total, 2) #round the amount
print("The total bill is $", total) #print output
```

Note: We're adding useful comments to our code (non-trivial lines)

Flow Control: Conditionals

if…elif…else

- Allows us to make decisions and execute “code blocks” (groups of statements) based on logical conditions
  - Code blocks are indented 4 spaces (single tab)

if…elif…else

- Allows us to make decisions and execute “code blocks” (groups of statements) based on logical conditions
  - Code blocks are indented 4 spaces (single tab)
- *if* condition is True, execute statements in body of *if*

```
x = int(input("2 + 2 ="))
if x != 4:
    print("… try again”)
```

if…elif…else

- Allows us to make decisions and execute “code blocks” (groups of statements) based on logical conditions
  - Code blocks are indented 4 spaces (single tab)
- *if* condition is True, execute statements in body of *if*

```
x = int(input("2 + 2 ="))
if x != 4:
    print("… try again”)
```

- *if* condition is True, execute statements in body of *if*, *else* (otherwise), execute statements in body of *else*

```
x = int(input("2 + 2 ="))
if x == 4:
    print("Basic arithmetic holds”)
else:
    print("...try again”)
```

if…elif…else

- *if* condition is True, execute statements in body of *if*,

  elif (else if) another condition is True, execute statements in body of *elif*,

  else (otherwise), execute statements in body of *else*

```
age = int(input("Enter your age:"))
if age < 100:
    print("In", 100 - age, "years you will be 100 years old")
elif age == 100:
    print("You are", age, "years old")
else:
    print("A century and going strong!")
```

if…elif…else

- You can have multiple *elif* conditions!

```
age = int(input("Enter your age:"))
if age == 0:
    print(”Seriously?")
elif age < 100:
    print("In", 100 - age, "years you will be 100 years old")
elif age == 100:
    print("You are", age, "years old")
else:
    print("A century and going strong!")
```

Importance of Code Block Indentation

- Prints ‘a’ and ‘b’ if x < 5

```
if x < 5:
    print(‘a’)
    print(‘b’)
```

Importance of Code Block Indentation

- Prints ‘a’ and ‘b’ if x < 5

```
if x < 5:
    print(‘a’)
    print(‘b’)
```

- Prints ‘a’ if x < 5, otherwise prints ‘b’

```
if x < 5:
    print(‘a’)
else:
    print(‘b’)
```

Importance of Code Block Indentation

- Prints ‘a’ and ‘b’ if x < 5

```
if x < 5:
    print(‘a’)
    print(‘b’)
```

- Prints ‘a’ if x < 5, and prints ‘b’

```
if x < 5:
    print(‘a’)
print(‘b’)
```

- Prints ‘a’ if x < 5, otherwise prints ‘b’

```
if x < 5:
    print(‘a’)
else:
    print(‘b’)
```

Importance of Code Block Indentation

- Prints ‘a’ and ‘b’ if x < 5

```
if x < 5:
    print(‘a’)
    print(‘b’)
```

- Prints ‘a’ if x < 5, and prints ‘b’

```
if x < 5:
    print(‘a’)
print(‘b’)
```

- Prints ‘a’ if x < 5, otherwise prints ‘b’

```
if x < 5:
    print(‘a’)
else:
    print(‘b’)
```

- Invalid syntax

```
if x < 5:
    print(‘a’)
print(‘b’) #needs to be indented
else:
    print(‘c’)
```

if…elif…else - Exercise

- Prompt the user for a numerical grade and print the appropriate letter grade
  - Get user input of a numerical grade
  - Convert the user input to an integer
  - Test the range of the number using flow control
    - Print the appropriate letter grade. For example, if the user enters a number between 90 – 100, give them an ‘A’.

if…elif…else - Exercise

- Prompt the user for a numerical grade and print the appropriate letter grade

```
grade = int(input('Enter your grade:'))
if grade >= 90:
    print('A’)
elif grade >= 80:
    print('B’)
elif grade >= 70:
    print('C’)
elif grade >= 60:
    print('D’)
else:
    print ('F’)
```

Multiple *if* Conditionals

- Ask the user to input an integer to evaluate

```
number = input('Please input an integer.’)
```

Multiple *if* Conditionals

- Ask the user to input an integer to evaluate

```
number = input('Please input an integer.’)
```

- It’s good practice to confirm an input is “numeric” before casting it to an int
  - Here, *isnumeric* checks if all characters in the string are numeric characters (digits)

```
if (number.isnumeric()):
    number = int(number)
    if (number > 20):
        print('Your input: ’ + str(number) + ’ > 20’)
    if (number > 10):
        print('Your input: ’ + str(number) + ’ > 10’)
    if (number > 0):
        print('Your input: ’ + str(number) + ’ > 0’)
else:
    print('Your input is not an integer or it’s negative.’)
```

- NOTE: *isnumeric* only works with positive integers

Catching Errors

- What if you don’t check for numeric input and cast anyway?
    - For example, what if your input is “twenty-two”?

Catching Errors

- What if you don’t check for numeric input and cast anyway?
  - For example, what if your input is “twenty-two”?
- You’ll get an error!
  - But you can catch errors like this

Catching Errors

- What if you don’t check for numeric input and cast anyway?
  - For example, what if your input is “twenty-two”?
- You’ll get an error!
  - But you can catch errors like this
- Here’s an even better way to check for an integer

```
number = input('Please input an integer.’)
#Try to cast the input
try:
    print(‘hello’)
    number = int(number)
    print(str(number) + “ is indeed an integer!”)
#Catch the raised exception if there is an error – i.e. it can’t be casted
except ValueError as e:
    print(“Your input is not an integer.”)
    print(e) #You can also access/print the exception’s message
```

Catching Errors

- Here's our full code asking the user to input an integer to evaluate

```
number = input('Please input an integer.')
try:
    number = int(number)
    if (number > 20):
        print('Your input: ' + str(number) + ' > 20')
    if (number > 10):
        print('Your input: ' + str(number) + ' > 10')
    if (number > 0):
        print('Your input: ' + str(number) + ' > 0')
    if (number < 0):
        print('Your input: ' + str(number) + ' is negative')
except ValueError as e:
    print("Your input is not an integer.")


#Note: The else clause is optional, and not really needed here
```

Homework 2

Homework 2

Will be assigned by tonight, Tuesday, January 23rd at midnight and due Tuesday, January 30th at midnight

- In this assignment, you will implement a **supermarket "game"** that allows the customer to shop for specific items
- The topics are:
  - Getting user input
  - Error checking
  - Variables & data types
  - Conditionals
- To complete the assignment:
  - Submit your completed *.py* file to Canvas