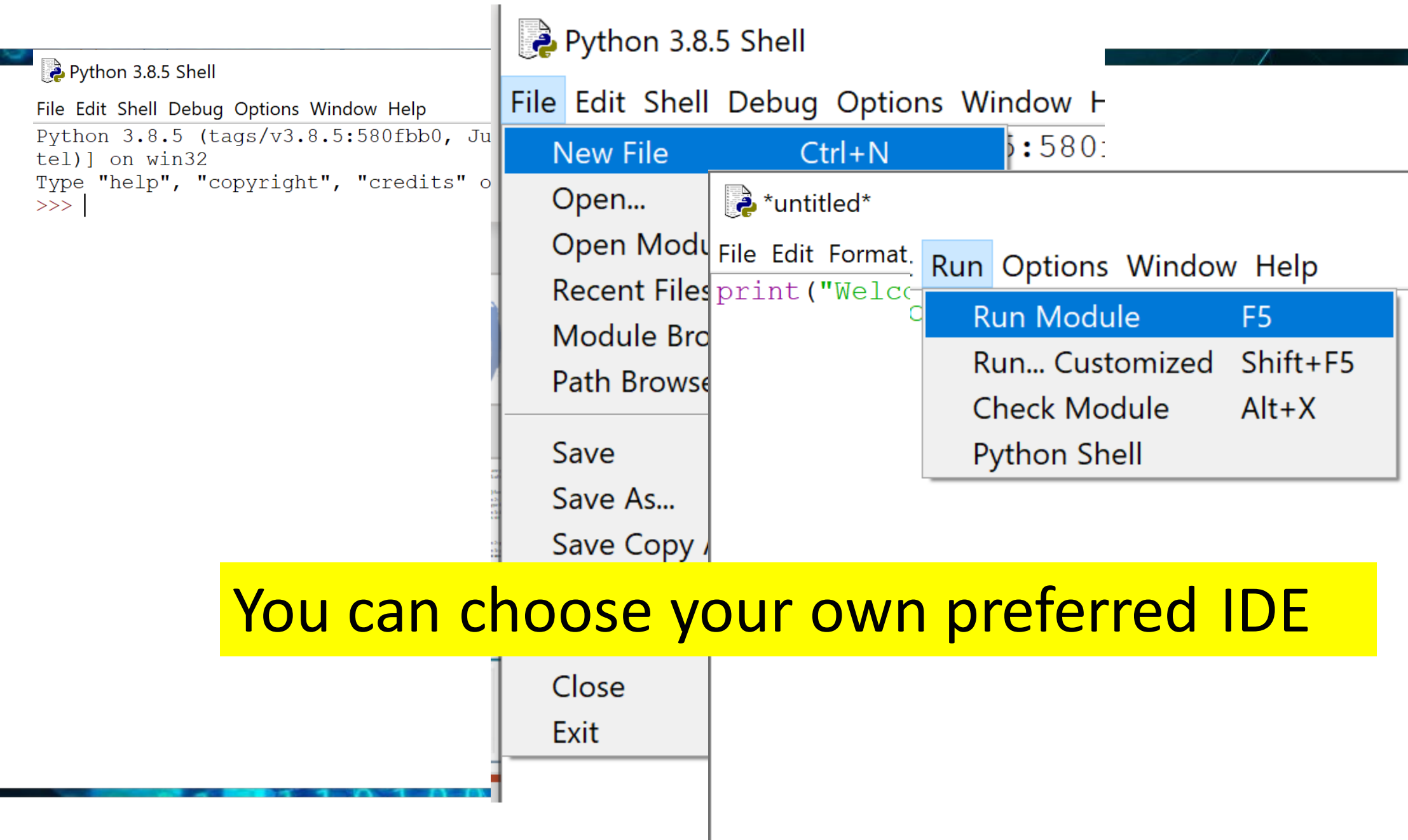




SC1003/CX1103 Introduction to computational thinking and programming

Python 2 vs Python 3

- There are plenty of differences between version 2 and version 3 of the Python language.
- **input()** function
- Python 2: evaluates the data it receives and returns the data type based on what it 'thinks' it should be.
- Python 3: **input()** is a function that always returns the data as str (i.e. string) data type object.
- **Print**
- Python 2: **print "Hello world"**
- Python 3: print() is a function and is written as **print ("Hello world")**
-



You can choose your own preferred IDE

A large, irregular blue ink splatter or watercolor blotch serves as the background for the title text. The splatter has a textured, painterly appearance with various shades of blue and white, creating a dynamic and artistic look.

Course Review

Data type, Variables

Identifier in Python

Identifier: a name given to an entity in Python

- Helps in differentiating one entity from another
- Name of the entity must be unique to be identified during the execution of the program



Rules for Writing Identifiers

What can be used?

- Uppercase and lowercase letters A through Z ($26 * 2 = 52$)
- The underscore, '_' (1)
- The digits 0 through 9, except for the first character (10)

$$52 + 1 + 10 = 63$$



Syntax Rules in Python

- Must begin with a letter or _

- 'Ab123' and '_b123' are ok
- '123ABC' is not allowed

- May contain letters, digits, and underscores

`this_is_an_identifier_123`

- Should **not** use keywords

- Upper case and lower case letters are different

'LengthOfRope' is **not** 'lengthofrope'

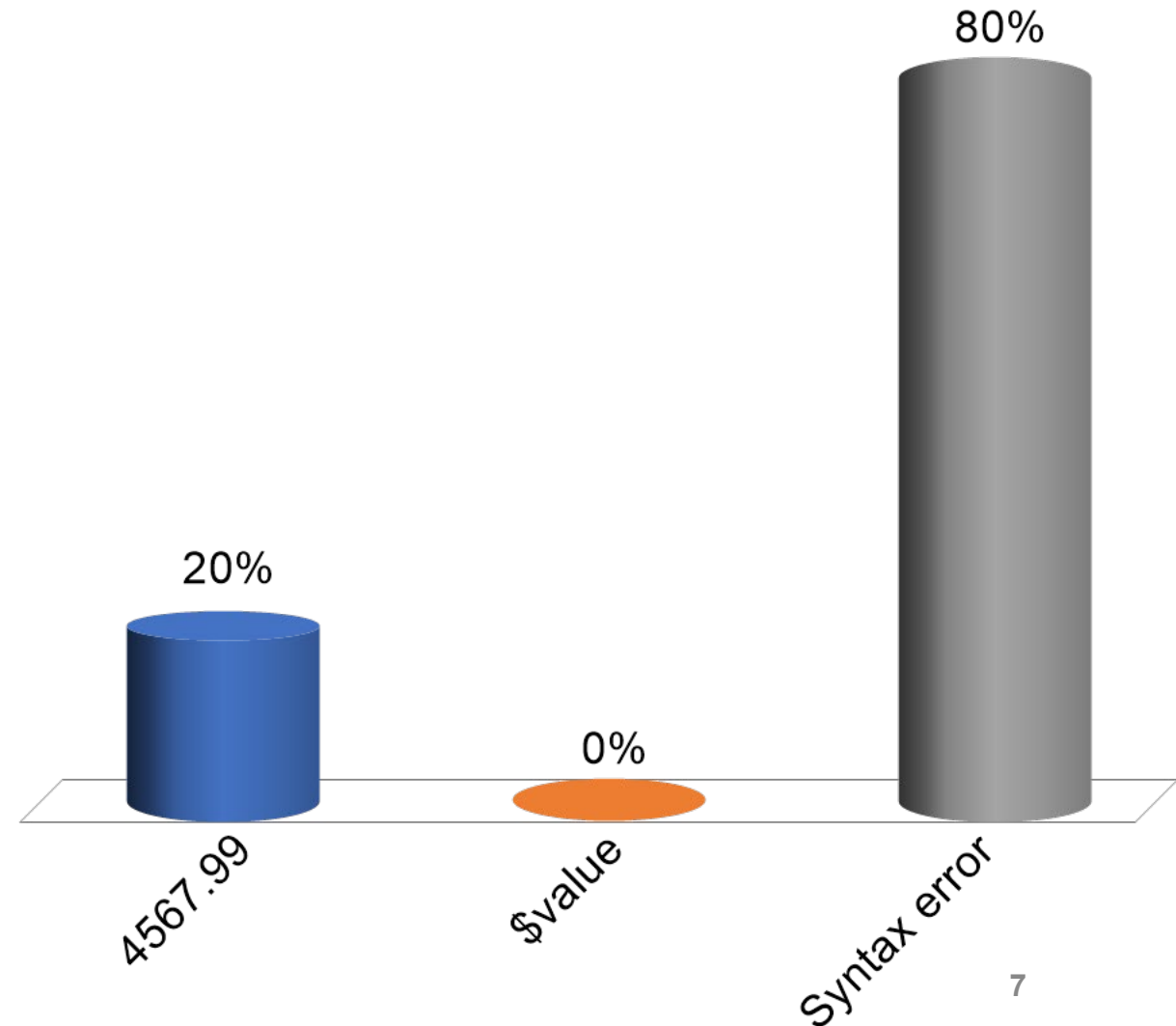
 *Python is **case sensitive***

- Can be of any length
- Names starting with _ have special meaning

What is the output of the following code?

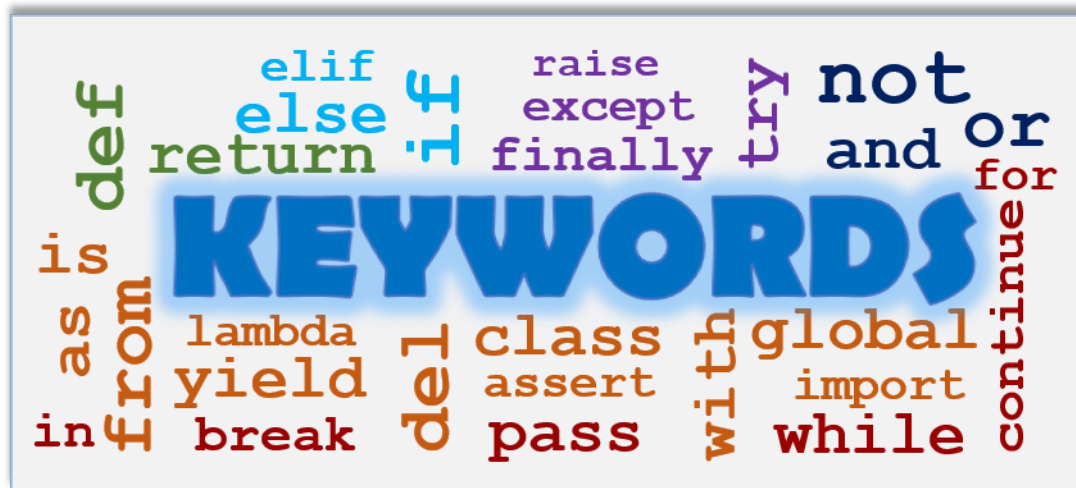
```
$value = 4567.99  
print($value)
```

- A. 4567.99
- B. \$value
- ✓ C. Syntax error





Keywords

- Special words reserved in Python
- Programmers should **not** use keywords to *name* things



Note: Old Python keyword '**exec**' was removed in Python 3


```
alice = 50  
peter = 70  
sum = alice + peter  
print (sum)  120
```

```
scores = [1,2,3,4]  
total = sum(scores)  
print (total) 
```

```
Traceback (most recent call last):  
  File "F:\LF work\LF running courses\CX1103\Lectures\test1.py", line 7, in <module>  
    total = sum(scores)  
TypeError: 'int' object is not callable
```

		Built-in Functions		
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

Python Naming Conventions

```
import math
radiusString = input("Enter the radius of your circle:")
radiusFloat = float (radiusString)
circumference = 2 * math.pi * radiusFloat
area = math.pi * radiusFloat * radiusFloat
```



VS.

```
import math
a = input("Enter the radius of your circle:")
b = float (a)
c = 2 * math.pi * b
d = math.pi * b * b
```



What is c? It is not immediately clear.

- Both programs work
 - They are different when **readability counts**
-
- variable names should be in lowercase, with words separated by underscores as necessary to improve readability
e.g. **radius_float**
 - mixedCase is allowed
e.g. **radiusFloat**

String - designated as **'str'**

- It is basically a sequence, typically a sequence of characters delimited by single quote ('...') or double quotes ("...") for a single line sentence, even triple quotes (""...") to display a paragraph with multiple lines
- First *collection type* that was discussed
- Collection type contains multiple objects organized as a single object



Examples

```
>>> a = 'Length'
>>> b = "8225 welcome"
>>> c = "ewwew sdcd &8 $5##"
>>> d = 'ewwew sdcd &8 $5##'
>>> e = '''

Welcome to Stock Master!

Please select a function to continue:

(1) Search for a stock code by
keyword

(2) View stock trend by code

(sample input 1 or 2)

'''
```

```
e = '''
Welcome to Stock Master!
Please select a function to continue:
(1) Search for a stock code by keyword
(2) View stock trend by code
(sample input 1 or 2)
'''

print(e)

a = 'Length'
print(a)
b= "8225 welcome"
print(b)
```



```
===== RESTART: D:/LF work/LF current Cc

Welcome to Stock Master!
Please select a function to continue:
(1) Search for a stock code by keyword
(2) View stock trend by code
(sample input 1 or 2)

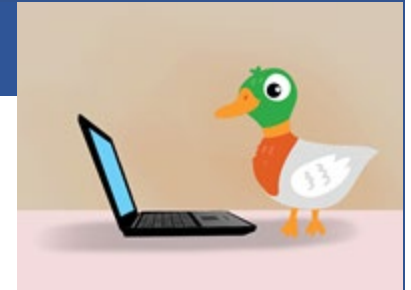
Length
8225 welcome
```



Compared to C and Java, how does Python know the data types?

Python uses *Duck-Typing*

“When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.” – James Whitcomb Riley



Examples

```
>>> a = 99
>>> b = 99.9
>>> c = '100'
>>> d = True
```



Four variables!




What are their data types?

Data Types (Cont'd)

Type Function

In Python, the **type()** function allows you to know the type of a **variable** or **literal**.



```
>>> x = 9
>>> type (x)
<class 'int'>
>>> x = 7.8
>>> type (x)
<class 'float'>
>>> x = "Welcome"
>>> type (x)
<class 'str'>
>>> x = 'Python'
>>> type (x)
<class 'str'>
>>> type (8.9)
<class 'float'>
```

- Python does not have variable declaration, like Java or C, to announce or create a variable.
- A variable is created **by just assigning a value to it** and the type of the value defines the type of the variable.
- If another value is re-assigned to the variable, **its type can change**.

Assignment Operator

```
import math
radiusString = input("Enter the radius of your circle:")
radiusFloat = float (radiusString)
circumference = 2 * math.pi * radiusFloat
area = math.pi * radiusFloat * radiusFloat
```

Basic Syntax

Left Hand Side (LHS) = Right Hand Side (RHS)



a variable



an expression

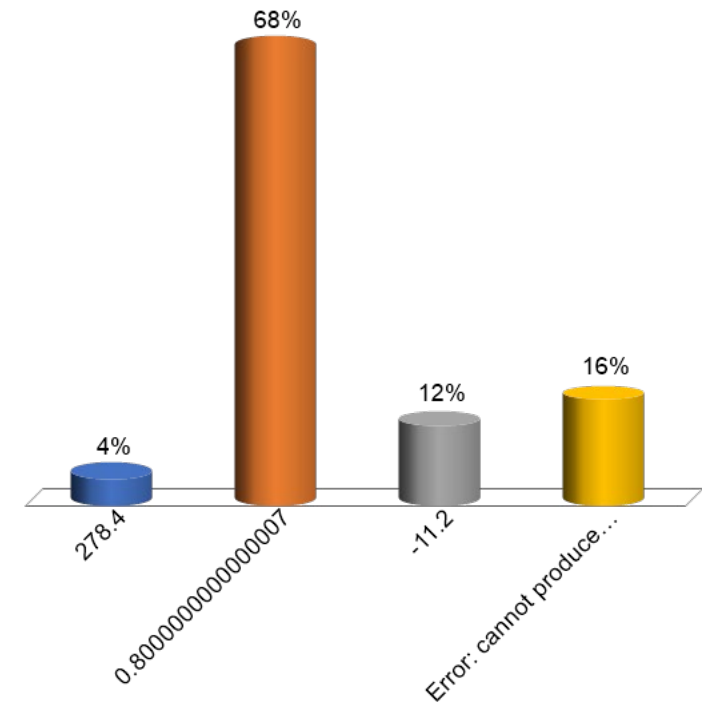
Assignment means:

1. Evaluate the expression on RHS
2. Take the resulting value and assign it to the name (variable) on the LHS.

What is the output of the following program?

```
pay_1 = 12
pay1 = 23.2
result1 = pay_1 * pay1
resultT_1 = pay_1*2 - pay1
result_1 = pay_1 - pay1
print(resultT_1)
```

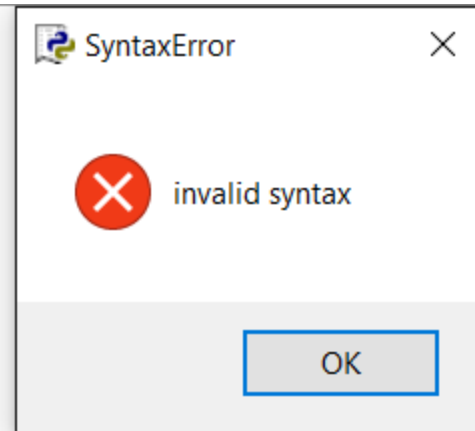
- A. 278.4
- ✓ B. 0.800000000000000000007
- C. -11.2
- D. Error: cannot produce output



$$0.8 = \frac{1}{2} + \frac{1}{4} + \frac{1}{32} + \dots$$

If you get syntax error as follows, which line of the code you should correct?

```
first_num = 6
second_num = 7
print (first_num
sum_num = first_num + second_sum
print (sun_num)
```



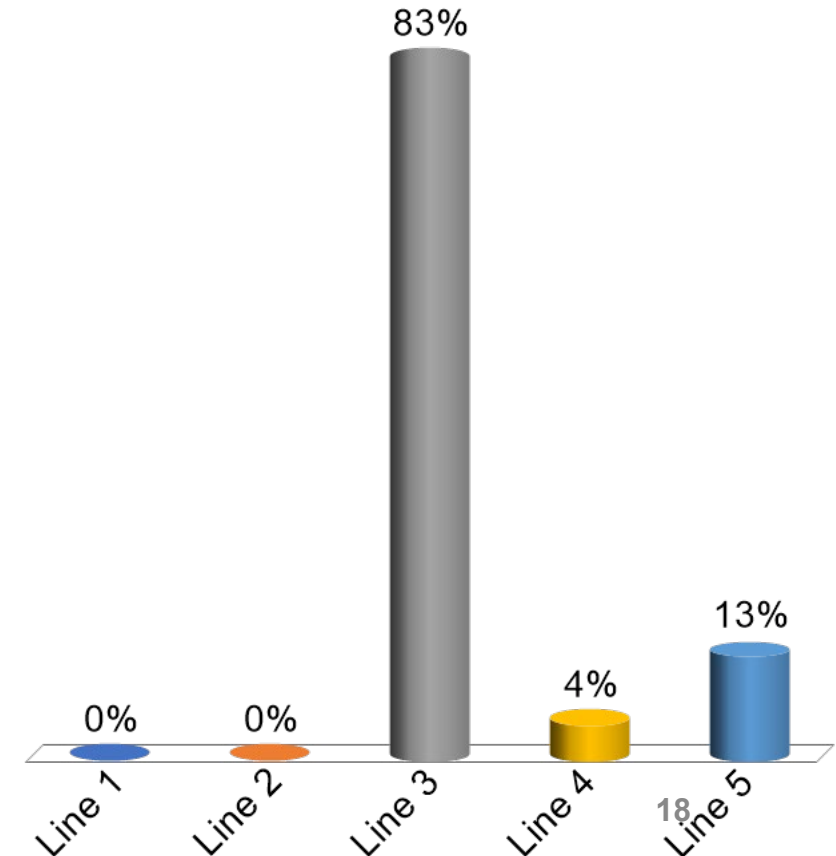
A. Line 1

B. Line 2

✓ C. Line 3

D. Line 4

E. Line 5





FUNCTION CALL



INPUT, OUTPUT



3. INPUT, PROCESSING, AND OUTPUT

- Typically, computer performs three-step process
 - Receive input
 - Input: any data that the program receives while it is running
 - Perform some process on the input
 - Example: mathematical calculation
 - Produce output

```
# 1. prompt user for the radius
# 2. apply circumference and area formulae
# 3. print the results

import math
radiusString = input("Enter the radius of your circle:")
radiusFloat = float (radiusString)
circumference = 2 * math.pi * radiusFloat
area = math.pi * radiusFloat * radiusFloat

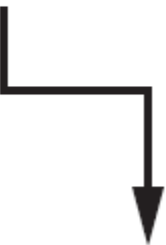
print()    # print a line break
print ("The circumference of your circle is:",circumference, ", and the area
is:", area)
```



Passing Arguments to Functions

- Argument: piece of data that is sent into a function
 - Function can use argument in calculations and processing
 - When calling the function, the argument is placed in parentheses following the function name

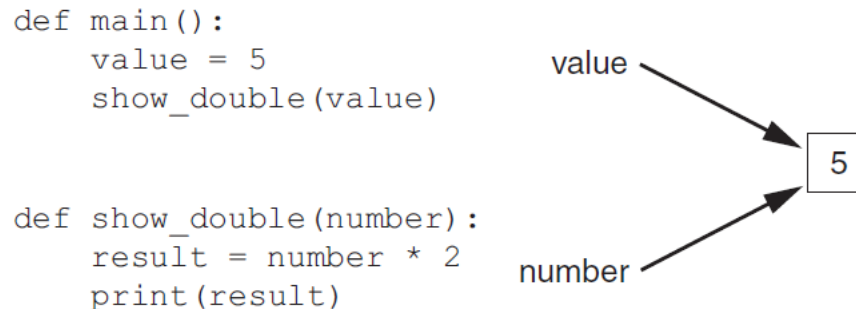
Figure 5-13 The `value` variable is passed as an argument

```
def main():  
    value = 5  
    show_double(value)  
      
  
def show_double(number):  
    result = number * 2  
    print(result)
```

Parameter variable in Functions

- Parameter variable: variable that is assigned the value of an argument when the function is called
 - The parameter and the argument reference the same value
 - General format:
 - `def function_name(parameter) :`
 - Scope of a parameter: the function in which the parameter is used

Figure 5-14 The `value` variable and the `number` parameter reference the same value



Reading Input from the Keyboard

- Most programs need to read input from the user
- Built-in `input` function reads input from keyboard
 - Returns the data as a string, even if the user enters numeric data.
 - Format: `variable = input(prompt)`
 - `prompt` is typically a string instructing user to enter a value
 - Does not automatically display a space after the prompt

```
math_str = input('What is your Math score?')
```



```
What is your Math score?81
```

Reading Numbers with the `input` Function

- `input` function always returns a string
- Built-in functions convert between data types
 - `int(item)` converts *item* to an `int`
 - `float(item)` converts *item* to a `float`
 - Type conversion only works if item is valid numeric value, otherwise, throws exception

```
math_str = input('What is your Math score? ')
math_float = float(math_str)
print(math_float)
```

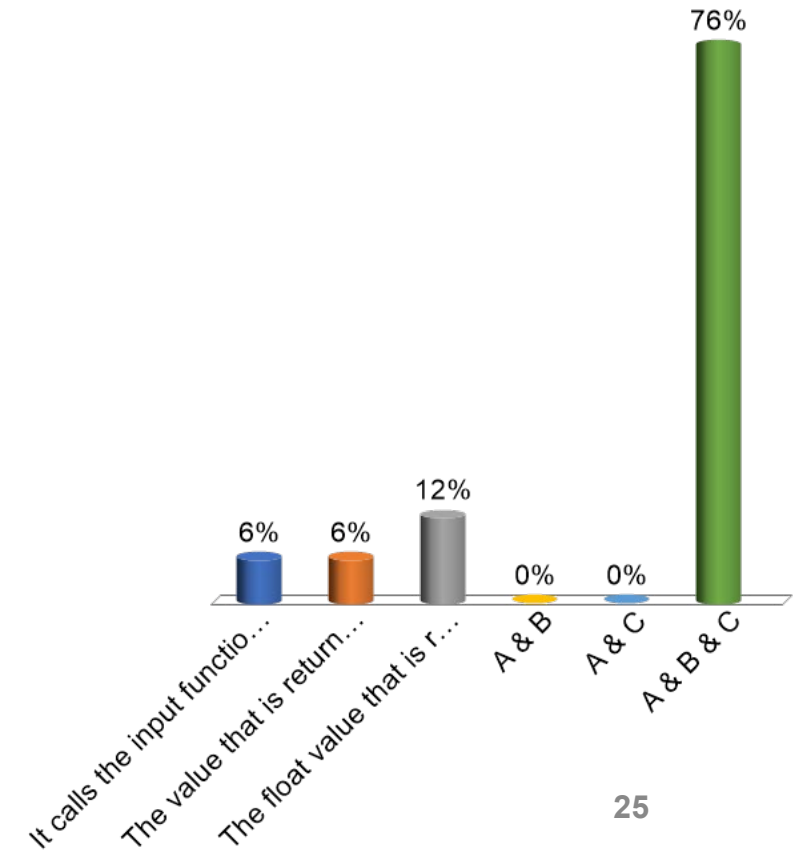


```
What is your Math score? 81
81.0
```


How the following code works?

```
math_score = float(input('What is your Math score?'))
```

- A. It calls the input function to get a value entered at the keyboard
- B. The value that is returned from the input function (a string) is passed as an argument to the float() function
- C. The float value that is returned from the float() function is assigned to the math_score variable
- D. A & B
- E. A & C
- ✓ F. A & B & C



Displaying Output with the `print` Function

```
print(  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

- `print` function: displays output on the screen
- Argument: data given to a function
 - Example: data that is printed to screen
- Statements in a program execute in the order that they appear
 - From top to bottom

```
print ("Hello")  
print ('Python')  
print ("1003")
```



```
Hello  
Python  
1003
```

Passing Multiple Arguments

Python allows writing a function that accepts multiple arguments

- Parameter list replaces single parameter
 - Parameter list items separated by comma

Arguments are passed *by position* to corresponding parameters

- First parameter receives value of first argument, second parameter receives value of second argument, etc.

Passing Multiple Arguments (cont'd.)

Figure 5-16 Two arguments passed to two parameters

```
def main():  
    print('The sum of 12 and 45 is')  
    show_sum(12, 45)
```

```
def show_sum(num1, num2):  
    result = num1 + num2  
    print(result)
```



Displaying Multiple Items with the `print` Function

- Python allows one to display multiple items with a single call to `print`
 - Items are separated by commas when passed as arguments
 - Arguments displayed in the order they are passed to the function
 - Items are automatically separated by a space when displayed on screen while using default `sep`

```
print(  
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
print("Hello", 'Python', "1003")  Hello Python 1003
```

Displaying Multiple Items with the `print` Function cont'

- `print` function uses space as item separator
 - Special argument `sep='delimiter'` causes `print` to use *delimiter* as item separator

```
print(  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

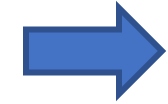
```
print("Hello", "Python", "1003", sep="#_#")
```



Hello#_#Python#_#1003

Escape Sequences

```
print ("Hello")  
print ('Python')  
print ("1003")
```



```
Hello  
Python  
1003
```

- The newline character `\n` is called an **escape sequence**

ESCAPE SEQUENCE	MEANING
<code>\b</code>	Backspace
<code>\n</code>	Newline
<code>\t</code>	Horizontal tab
<code>\\</code>	The <code>\</code> character
<code>\'</code>	Single quotation mark
<code>\"</code>	Double quotation mark

[TABLE 2.3] Some escape sequences in Python

Special characters appearing in string literal

Preceded by backslash (`\`)

Examples:
newline (`\n`),
horizontal tab (`\t`)

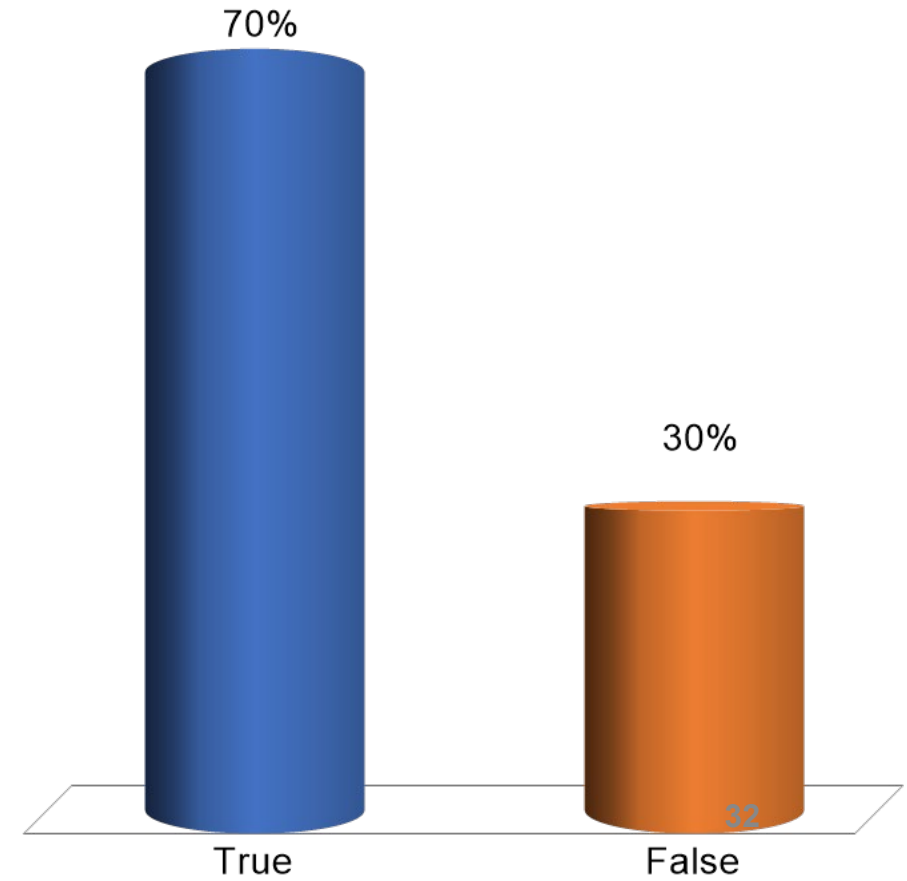
Treated as commands embedded in string

Is the output of the following code correct?

```
print ("Hello\n", 'Python\t', "1003")
```

```
Hello
Python  1003
```

- ✓ A. True
- B. False




argument `end` in `print` function

- `print` function displays line of output
 - Newline character at end of printed data
 - Special argument `end='delimiter'` causes `print` to place *delimiter* at end of data instead of newline character

```
print(  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
print ("Hello",end = '*')  
print ('Python',end = '@')  
print ("1003")
```



```
Hello*Python@1003
```

What is the output of the following code?

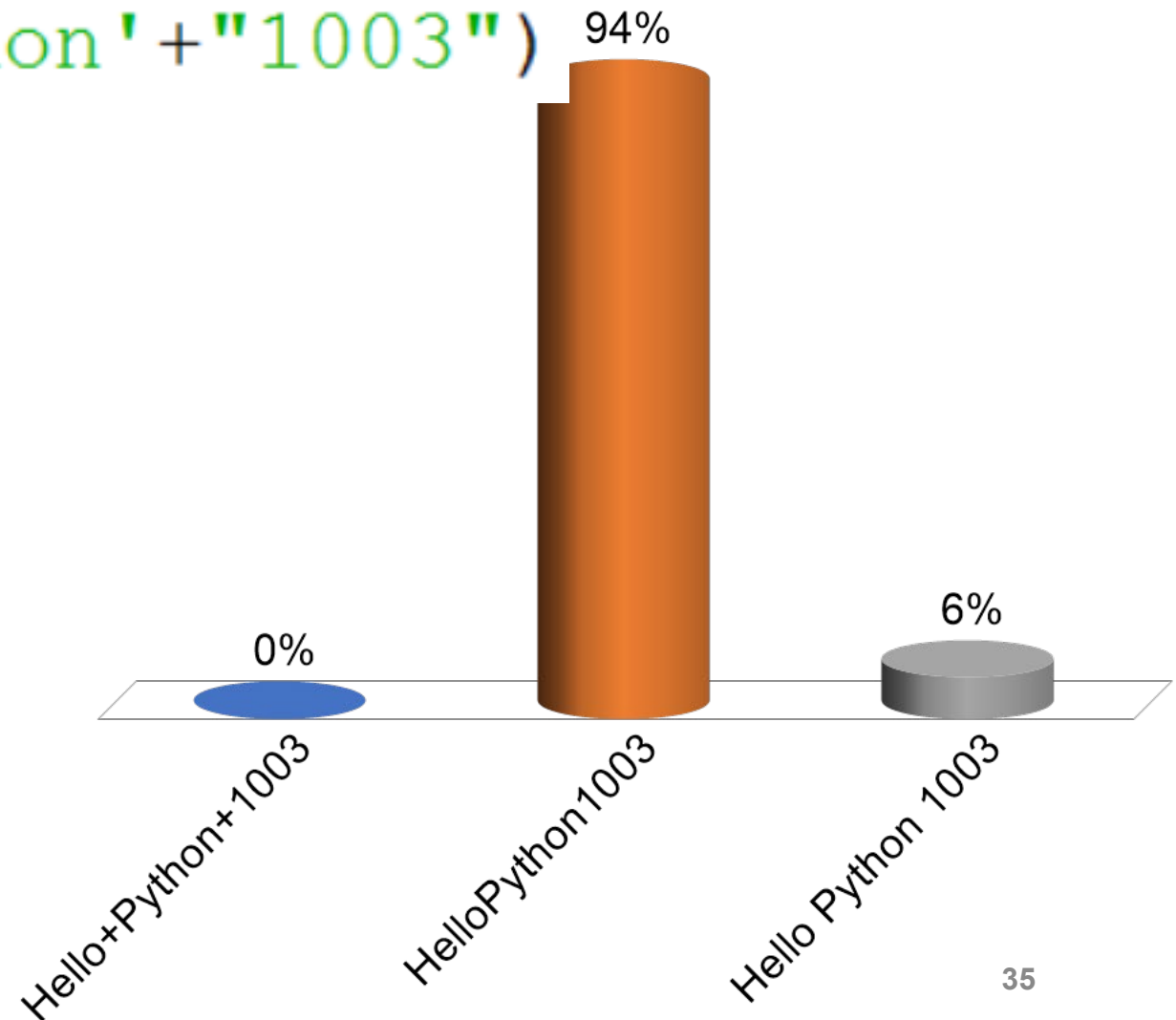
```
print("Hello", "Python", "1003", sep = "~", end = "##")  
print("Week4", "Lecture")
```

- A. Hello##Python##1003##Week4##Lecture
- ✓ B. Hello~Python~1003##Week4 Lecture
- C. Hello~Python~1003##Week4~Lecture
- D. Hello~Python~1003##Week4 Lecture##
- E. Hello~Python~1003##
Week4 Lecture

What is the output of the following code?

```
print ("Hello" + 'Python' + "1003")
```

- A. Hello+Python+1003
- ✓ B. HelloPython1003
- C. Hello Python 1003



+ operator

+ operator for two numbers: Add two numbers

+ operator for two strings: String concatenation.

It appends one string to another.

Standard Library Functions and the `import` Statement

```
import math
radiusString = input("Enter the radius of your circle:")
radiusFloat = float (radiusString)
circumference = 2 * math.pi * radiusFloat
area = math.pi * radiusFloat * radiusFloat
```

- Standard library: library of pre-written functions that comes with Python
 - *Library functions* perform tasks that programmers commonly need
 - Example: `print`, `input`, `range`, `math.sin`
 - Viewed by programmers as a “black box”
- Some library functions built into Python interpreter
 - To use, just call the function

		Built-in Functions		
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

Given the description of the built-in function round as follows:

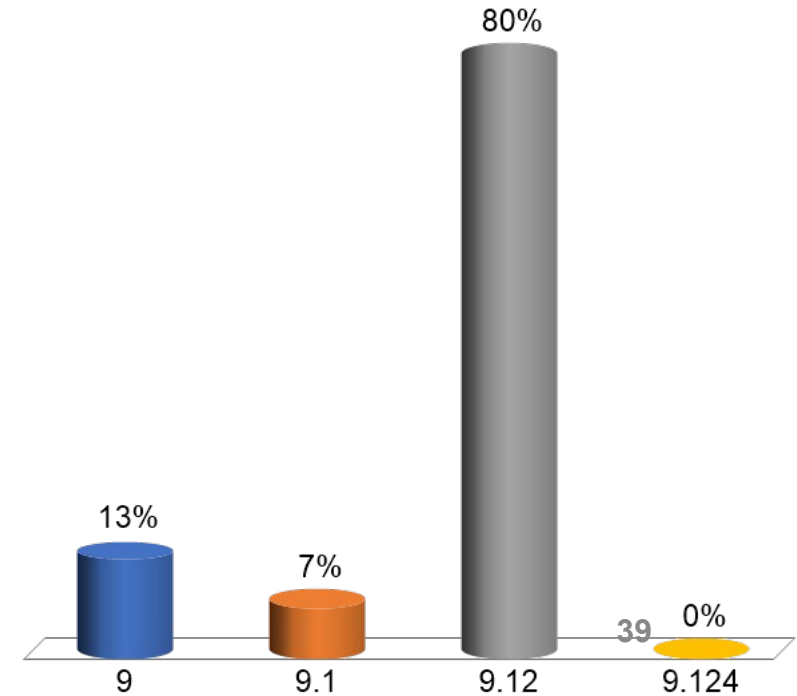
round(*number*[, *ndigits*])

Return *number* rounded to *ndigits* precision after the decimal point. If *ndigits* is omitted or is `None`, it returns the nearest integer to its input.

What is the output of the following code?

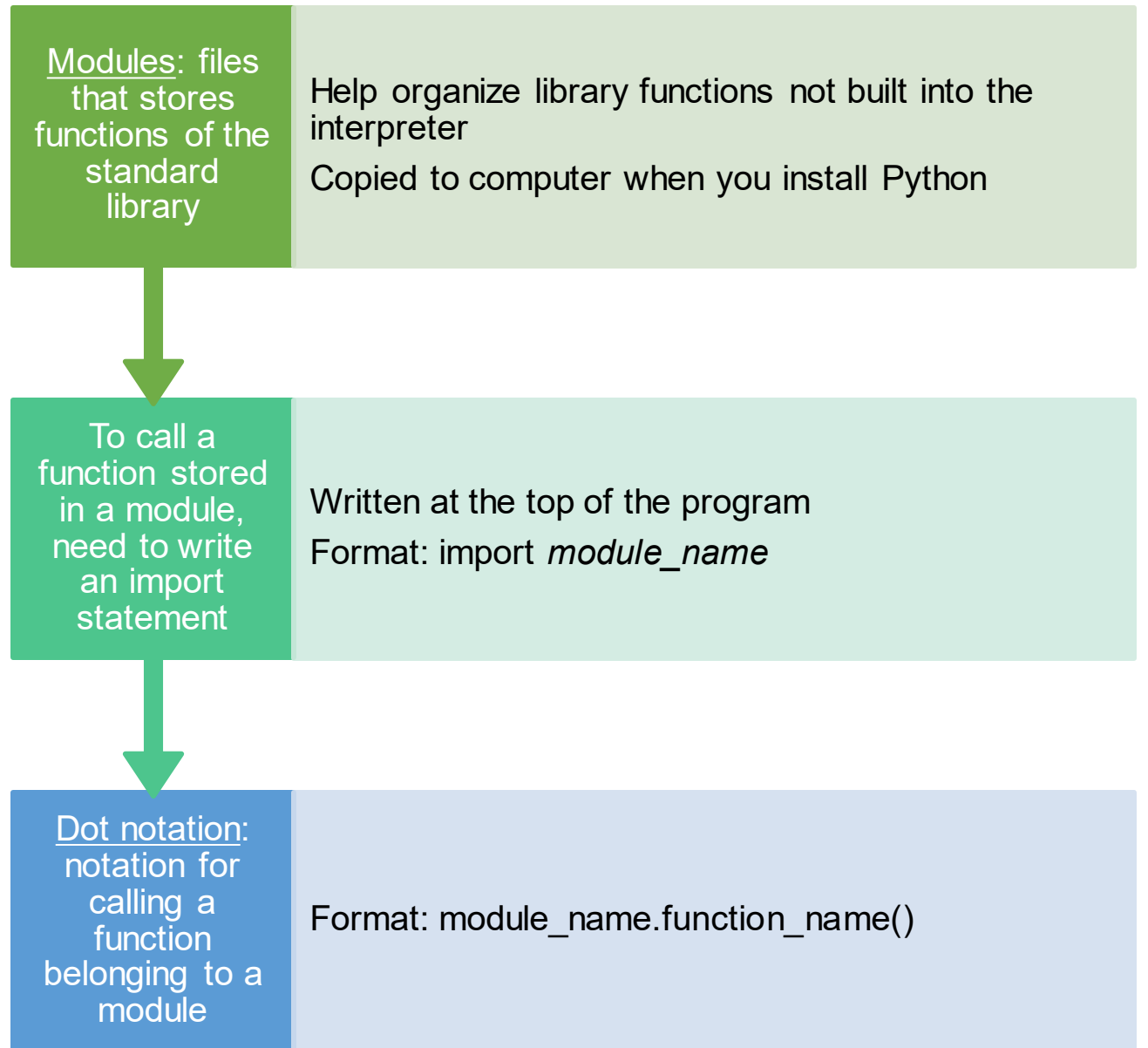
```
print(round(9.1235, 2))
```

- A. 9
- B. 9.1
- ✓ C. 9.12
- D. 9.124





Standard Library Functions and the `import` Statement (cont'd.)



The math Module

- math module: part of standard library that contains functions that are useful for performing mathematical calculations
 - Typically accept one or more values as arguments, perform mathematical operation, and return the result
 - Use of module requires an `import math` statement

The `math` Module (cont'd.)

Table 5-2 Many of the functions in the `math` module

<code>math</code> Module Function	Description
<code>acos(x)</code>	Returns the arc cosine of <code>x</code> , in radians.
<code>asin(x)</code>	Returns the arc sine of <code>x</code> , in radians.
<code>atan(x)</code>	Returns the arc tangent of <code>x</code> , in radians.
<code>ceil(x)</code>	Returns the smallest integer that is greater than or equal to <code>x</code> .
<code>cos(x)</code>	Returns the cosine of <code>x</code> in radians.
<code>degrees(x)</code>	Assuming <code>x</code> is an angle in radians, the function returns the angle converted to degrees.
<code>exp(x)</code>	Returns e^x
<code>floor(x)</code>	Returns the largest integer that is less than or equal to <code>x</code> .
<code>hypot(x, y)</code>	Returns the length of a hypotenuse that extends from (0, 0) to (<code>x</code> , <code>y</code>).
<code>log(x)</code>	Returns the natural logarithm of <code>x</code> .
<code>log10(x)</code>	Returns the base-10 logarithm of <code>x</code> .
<code>radians(x)</code>	Assuming <code>x</code> is an angle in degrees, the function returns the angle converted to radians.
<code>sin(x)</code>	Returns the sine of <code>x</code> in radians.
<code>sqrt(x)</code>	Returns the square root of <code>x</code> .
<code>tan(x)</code>	Returns the tangent of <code>x</code> in radians.

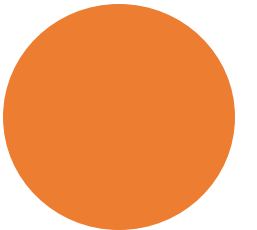
The import Statement

- import <module_name>
 - module_name.functionName()

```
import math  
y = math.log(4)
```

- from <module_name> import <functionName(s)>
 - functionName()

```
>>> from sense_hat import SenseHat  
>>> sense = SenseHat()  
>>> sense.show_message("Hello NTU")
```



The `math` Module (cont'd.)

The `math` module defines variables `pi` and `e`, which are assigned the mathematical values for π and e

- Can be used in equations that require these values, to get more accurate results

Variables must also be called using the dot notation

- Example:
 - `circle_area = math.pi * radius**2`