COMP 1023 Introduction to Python Programming

Basic Programming Language Elements

Dr. Cecia Chan, Prof. SC Cheung, Dr. Alex Lam, Dr. Desmond Tsoi

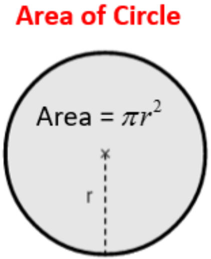
Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

Part I

Python Fundamentals

Writing a Simple Python Program

```python
'''
A simple program to set the circle's radius to 20, compute the area using the formula:
area =  PI * radius * radius, and display the result on the screen.
'''
# Filename: compute_area.py

def main():
    # Define the radius of the circle
    radius = 20

    # Define PI as 3.14159
    PI = 3.14159

    # Compute area
    area = PI * radius * radius

    # Display results
    print("The area for the circle of radius", radius, "is", area)

if __name__ == "__main__":
    main()
```



Output: `The area for the circle of radius 20 is 1256.636`

Comments

- Adding comments to a program increases its readability.
- Comments are NOT treated as computer instructions; they are only for humans to read, not for computers.
- In Python, there are two ways to add comments:

Syntax

1. The line after a hash symbol # is treated as a comment.

```
# Single line comment
```

2. All the lines between triple quotes (''' or """) are treated as comments:

```
'''
Multiple line comment
Comment line 1
...
Comment line n
'''
```

main()

- main() is the entry point of a Python program or script.
- It serves as the designated function where the core logic of the program resides.
- It is good practice to define a main function in Python (although it is not mandatory).

Syntax

```
def main():  # def means define
    # Statement(s)
    # ...

if __name__ == "__main__":
    main()
```

- When running the program (e.g., `python my_program.py`), Python executes the code from top to bottom.
- Upon reaching the line `if __name__ == "__main__":`, Python checks the value of `__name__`. If `__name__` equals `"__main__"`, it calls the `main()` function, executing the code inside it.

Comparison: With main() vs. Without main()

```python
# Filename: compute_area1.py

def main():
    radius = 20   # radius is now 20
    PI = 3.14159  # PI is 3.14159
    area = PI * radius * radius
    print("The area for the circle of radius", radius, "is", area)

if __name__ == "__main__":
    main()
```

```python
# Filename: compute_area2.py

radius = 20   # radius is now 20
PI = 3.14159  # PI is 3.14159
area = PI * radius * radius
print("The area for the circle of radius", radius, "is", area)
```

- You will not see a difference when executing these two programs using the commands: `python compute_area1.py` and `python compute_area2.py`.
- However, differences will become apparent in specific scenarios later!

Writing a Python Program

- Once you have defined the main function, you can start instructing the computer to do something!
- Here, "something" includes:
  1. Typical statements:
     - Object creation and initialization, assignments, function calls.
  2. Selective/Branching statements, i.e., statements for selecting which statements to execute:
     - `if..else`, `match..case`.
  3. Looping/Iterative statements, i.e., statements for repeatedly executing code:
     - `for`, `while`.

We will cover all these topics one by one in this course!

Part II

Basic Elements of Python Programming

Basic Data Types, Variables & Literals

- Pre-defined values or data entered by a user (via keyboard/mouse) are typically stored in main memory by computer programs.
- Python programs manage this through elements called objects.
- By creating objects in a program, data of different types can be stored in the computer's memory.
- To give an object a name, we define a variable, which holds a reference to that object.
- Before discussing how to create objects and declare variables, we will first introduce:
  - 6 basic data types and 5 container data types
  - 6 basic literal types

  supported by Python.

Data Types

- Python has 6 basic types, which are integers, floats, complexes, booleans, strings, None, and 5 container types, including lists, tuples, dictionaries, sets, frozenset.

Name	Type	Description
Integers	int	Whole numbers, e.g., 3, 300, 200
Floating points	float	Numbers with a decimal point: 2.3, 4.6, 100.0, 236., 2.23e-3
Complex numbers	complex	Complex numbers with real and imaginary parts, e.g., 1 + 2j
Booleans	bool	Logical values indicating True or False
Strings	str	Ordered sequences of characters: `"Hello"`, `‘Desmond’`, `"2000"`
None	None	Represents the absence of a value or a null value (e.g., None)
Lists	list	Ordered sequences of objects: [10, `"Hello"`, 200.3]
Tuples	tuple	Ordered immutable sequences of objects: (`"a"`, `"b"`)
Dictionaries	dict	Ordered* Key:Value pairs: {`"mykey"` : `"value"`, `"name"` : `"Desmond"`}
Sets	set	Unordered collections of unique objects: {`"a"`, `"b"`}
Frozen Sets	frozenset	Immutable version of set: `frozenset(["a", "b", "c"])`

*Starting from Python 3.7, dictionaries are ordered.

type() method returns the class type of the argument (object) passed as a parameter. The `type()` function is mostly used for debugging purposes.

Syntax

```
type(<object>)
```

6 Basic Literal Types

- Literals are values associated with data types used in a program.
- In Python, there are 6 basic literal types:
  1. Integer literals:
     - 42, -3, 18, 20493, 0
  2. Floating point literals:
     - 7.35, -19.93423, 42.0
  3. Complex literals:
     - 1 + 2j, 3 + 4j
  4. Boolean literals:
     - True, False
  5. String literals:
     - `'a'`, `'a++'`, `'X'`, `'Desmond'`, `"Cecia"`, `"Alex"`
  6. None literal:
     - None

Note

In Python, the letter `j` is used to denote the imaginary unit instead of `i` to avoid confusion with electrical engineering, where `i` commonly represents current. This choice ensures clarity in mathematical computations involving complex numbers, particularly in fields like engineering and physics.

True or False

- Python uses 1 to represent True and 0 for False.
- You can use explicit conversion functions to convert between types:
  - The `int()` function converts a Boolean value to an integer.
  - The `bool()` function converts a numeric value to a Boolean value.
- Non-zero values are treated as True.
- Zero is treated as False.
- The `bool()` function returns False if the value is 0; otherwise, it returns True.

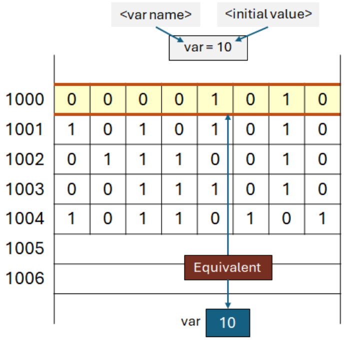
```python
print(int(True))    # Print 1
print(int(False))   # Print 0
print(bool(0))      # Print False
print(bool(4))      # Print True
```

Strings

- Strings in Python can be enclosed in single quotes or double quotes (and in fact, triple quotes).
- For this course, we will follow this convention:
  - A string with one character will use single quotes (i.e., `''`).
  - A string with more than one character will use double quotes (i.e., `""`).

Object Creation and References

- Data of different types can be stored in the computer's memory.
- An object is a memory location with a type that stores a value. (You can think of an object as a box that can hold a single data value, but also complex data types that can contain multiple values or structures.)
- An object is created the moment we first assign a value to it.
- A variable must be defined to reference the object, making it accessible.



Syntax

```
<var name> = <initial value>
<var name1> = <var name2> = <initial value>
<var name1>, <var name2>, ... = <initial value1>, <initial value2>, ...
```

where `<var name>` or `<var nameN>` is the name of the variable in the program, which refers to the location in memory, and `<initial value>` or `<var nameN>` is the initially assigned value to the object (this process is called initialization!).

Rules for Choosing Variable Names

Python imposes the following rules for choosing variable names (or identifiers):

1. Must start with a letter from the Unicode character set (i.e., any letter from any alphabet, depending on the language's specifications. E.g., for English, a-z, A-Z) or an underscore (_). It cannot start with a digit.
2. The other characters can only be letters (including Unicode letters) (E.g., for English, a-z, A-Z), digits (0-9), and underscores (_).
3. Cannot contain a space or any other characters such as a dot (.), a question mark (?), an "at" sign (@) or a hyphen (-).
4. Does NOT have any length limit.
5. Furthermore, some words are prohibited from being used as variable names; we call these words reserved words.

Identifiers

Identifiers are names that identify elements such as variables and functions (to be introduced later) in a program.

List of Reserved Words

- A variable name cannot be any of the keywords in Python (known as reserved keywords), because keywords have special meanings to the Python interpreter.

Reserved Keywords			
and	del	if	pass
as	elif	import	raise
assert	else	in	return
async	except	is	True
await	False	lambda	try
break	finally	None	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	

Variable Name Examples

- Examples of valid variable names:
  - Area, Length, A, seven_And_1, my_program, desmond, GoOd, bAd.
- Examples of invalid variable names:
  - my-program (Hyphen is not allowed.)
  - Python.good (Dot sign is not allowed.)
  - 7and11 (Variable name cannot start with a digit.)
  - good or bad (Space is not allowed.)
  - c?0 (? is not allowed.)
  - A/C (/ is not allowed.)
  - pass (pass is not allowed as it is a reserved keyword.)

Example: Object Creation and Reference

```python
# Filename: object_creation_reference_demo1.py
# An example demonstrating how to create objects & reference them using variables in Python.

def main():
    my_age = 18      # Create an object with value 18 and reference it with my_age

    # my_id is not defined, which will cause an error if uncommented
    # my_id          # Uncommenting this line will raise a NameError

    # Create objects with values 1023, 2011, and 2012
    course1, course2, course3 = 1023, 2011, 2012
    print(my_age)    # Print 18
    # print(my_id)   # Uncommenting this line will raise a NameError

    # Summary: my_age, course1, course2, and course3 are references to their
    #          respective objects.

if __name__ == "__main__":
   main()
```

Example: Object Creation and Reference

```python
# Filename: object_creation_reference_demo2.py
# An example demonstrating how to create objects & reference them using variables in Python.

def main():
    my_age = 18                             # An integer object that stores 18
    my_height = 1.72                        # A float object that stores 1.72
    my_name, my_grade = "Desmond", 'A'      # Two string objects storing "Desmond" and 'A'
    good_guy, bad_guy = True, False         # Two boolean objects storing True and False

    print(my_age)                           # Print 18
    print(my_height)                        # Print 1.72
    print(my_name, my_grade)                # Print Desmond A
    print(good_guy, bad_guy)                # Print True False

    # Summary: my_age, my_height, my_name, my_grade, good_guy, and bad_guy
    # are references to their respective objects in memory.

if __name__ == "__main__":
   main()
```

Dynamic Typing in Python

- Python is dynamically typed, meaning that variables can refer to any type of objects.
- Advantages:
  - Flexibility: Variables can change types.
  - Ease of use: Simplifies code writing.
- Disadvantages:
  - Runtime errors: Type-related errors may only appear when the code is executed.
  - Less performance optimization compared to statically typed languages.
- Example:

```python
x = 10         # x is an integer
x = "Hello"    # x is now a string
```

To enhance the code quality, maintainability, and collaboration, it is a good practice to do type hinting.

Type Hinting

- To specify the type of a variable using a colon (:), followed by the type name.

Syntax

`<var name>: <type name> = <initial value>`

where `<var name>` is the name of the variable, `<type name>` is the type of the variable, and `<initial value>` is the value assigned to the variable (this is called initialization).

```
name: str = "Desmond"
gender: str = 'M'
age: int = 18
height: float = 1.72
is_good_guy: bool = True
```

Type hinting in Python does not enforce the type of data to be assigned. Tools like `mypy` can analyze your code and check for type consistency based on the hints provided.

Notes about Variable Declaration

- As Python is a case-sensitive language, it treats variables differently based on the use of uppercase and lowercase letters.
    - For example, the variable names “my_var” and “My_Var” are treated as two different variables.
- Meaningful variable names make programs more readable.
    - For example, “tax” is certainly better than “t”.

Named Constants

- A named constant (or simply a constant) represents permanent data that never changes.
- In our `compute_area` example, PI is a constant, which we represent using the descriptive name `PI` for its value.
- Python does not have a special syntax for naming constants.
- We can create a variable that uses all uppercase letters to name a constant, such as PI. However, it is still mutable (meaning it can be changed), but it is recommended not to do so.

Benefits

1. We don't have to repeatedly type the same value if it is used multiple times.
2. If you need to change the constant's value (e.g., from 3.14 to 3.14159 for PI), you only need to change it in a single location in the source code.
3. Descriptive names make the program easier to read.

Explicit Type Conversions

We can explicitly convert one data type to another.

- For example, we can convert:
  - a float value to an integer value and vice versa.
  - a string value to an integer value and vice versa.

```
val1 = 20
print(val1, type(val1))
print(float(val1), type(float(val1)), '\n')
val2 = 52.7
print(val2, type(val2))
print(int(val2), type(int(val2)), '\n')
val3 = "72"
print(val3, type(val3))
print(int(val3), type(int(val3)), '\n')
val4 = 86
print(val4, type(val4))
print(str(val4), type(str(val4)))
```

Output:

```
20 <class 'int'>
20.0 <class 'float'>

52.7 <class 'float'>
52 <class 'int'>

72 <class 'str'>
72 <class 'int'>

86 <class 'int'>
86 <class 'str'>
```

Python Convention & Good Programming Style

- Use meaningful names.
- Variable names
  - Use lowercase letters for variables with a single word.
  - For names consisting of several words, use underscores (i.e., _) to separate them. For example: `my_variable`, `total_sum`.
    (We call this snake case!)
- Named constants
  - Use uppercase letters for named constants (e.g., PI), and underscores to separate words (e.g., MAX_LIMIT).
- Source filenames
  - Use lowercase letters
  - For names consisting of several words, use underscores (i.e., _) to separate them. For example: `my_program.py`, `data_analysis.py`

Following these conventions helps keep your code organized and easily identifiable with the Python Community.

Mutability of Basic Types in Python

- Mutable Types:
  - Lists: Elements can be added, removed, or modified.
  - Dictionaries: Key-value pairs can be added, updated, or removed.
  - Sets: Elements can be added or removed.
- Immutable Types:
  - Integers: Any change or operation creates a new integer.
  - Floats: Any change or operation creates a new float.
  - Complex Numbers: Any change or operation creates a new complex number.
  - Booleans: Any change or operation creates a new Boolean.
  - Strings: Any change results in a new string.
  - Tuples: Once created, their elements cannot be changed or modified. However, if a tuple contains mutable objects, those objects can be modified.
  - Frozen Sets: Immutable version of set.
  - None: It represents a null value and cannot be changed.

Understanding mutability is crucial for effective data manipulation and memory management in Python.

Random Number Generation

- Python provides the `random` module for random number generation.
- Common functions include:
  - `random.random()` - Returns a float between 0.0 and 1.0.
  - `random.randint(a, b)` - Returns a random integer between `a` and `b` (inclusive).
  - `random.choice(sequence)` - Returns a random element from a non-empty sequence.
  - `random.sample(population, k)` - Returns a list of `k` unique elements from the population.
- Using Seed:
  - The `random.seed(seed)` function initializes the random number generator.
  - Setting the seed ensures that you get the same sequence of random numbers every time you run the code.
  - This is useful for debugging and reproducibility in experiments.
  - If you do not set a seed, Python will use the current system time or a random source from the operating system to initialize the random number generator, which results in different sequences of random numbers each time the program runs.

Random Number Generation

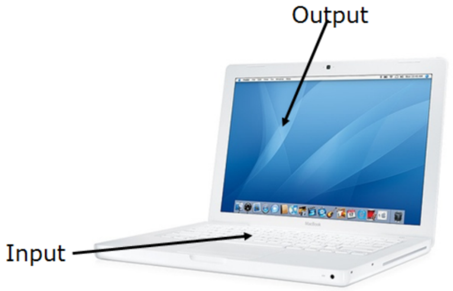
```
# Filename: random_number_generation.py
import random

def main():
    random.seed(42)  # Set the seed for reproducibility

    # Generate random numbers
    print(random.random())                    # Random float between 0.0 and 1.0
    print(random.randint(1, 10))              # Random integer between 1 and 10
    print(random.choice(['apple', 'banana', 'cherry']))  # Random choice from a list
    print(random.sample(range(100), 5))       # Sample 5 unique numbers from 0 to 99

if __name__ == "__main__":
    main()
```

Output:

```
0.6394267984578837
1
cherry
[35, 31, 28, 17, 94]
```

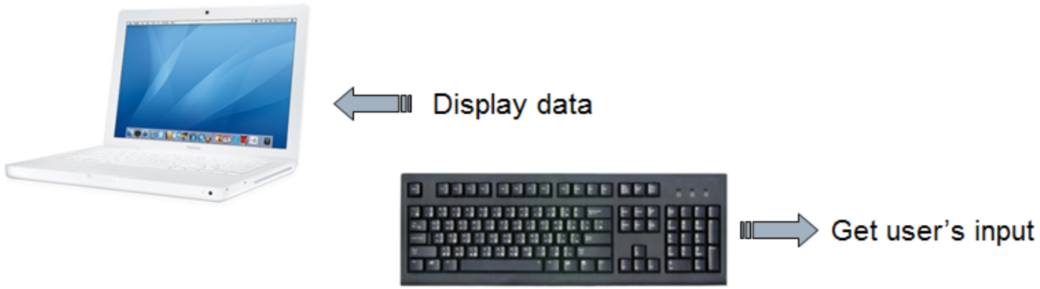
Part III

Basic Python Input and Output



Basic Python Input and Output (I/O)

- Two fundamental elements in a Python program are:
  1. Outputting data to the standard output device (the monitor's screen)
  2. Getting user input from the standard input device (the keyboard)



Output Data to the Standard Output Device

- The print() function is used to output data to the standard output device (screen).

Syntax

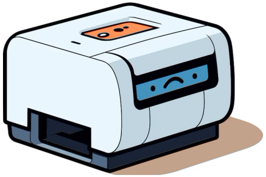
```
print(*<objects>, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- Parameters:
  - <objects>: The value(s) to be printed. * means it allows you to pass any number of arguments.
  - sep: The separator used between the values, defaulting to a space character.
  - end: After all values are printed, this is printed, defaulting to a new line.
  - file: The object where the values are printed; its default value is `sys.stdout` (screen).
  - flush: To ensure immediate output when `print()` is called, set `flush` to `True`.

Output formatting can be done using formatted strings (f-strings). You can include variables directly within the string by placing them inside curly braces . This allows for a more readable and concise way to format strings without needing to specify the order or use keyword arguments.

Output Data: Use of the end Parameter

```python
# Filename: use_of_end.py

def main():
    course_code = 1023
    print(course_code)
    print("You should be able to see this text", end=' ')
    print("on screen")
    print("How about this text?", end=' ')
    print("Can you see the effect?", end=' ')

if __name__ == "__main__":
    main()
```



Output:

```
1023
You should be able to see this text on screen
How about this text? Can you see the effect?
```

Output Data: Use of sep and end Parameters

```python
# Filename: use_of_sep_end.py

def main():
    print("Which is the best COMP course? :D")  # Print a string

    print(1, 0, 2, 3)
    print(1, 0, 2, 3, sep='-')
    print(1, 0, 2, 3, sep='~', end='*')
    print()  # Print '\n', i.e., move to the next line

if __name__ == "__main__":
    main()
```

Output:

```
Which is the best COMP course? :D
1 0 2 3
1-0-2-3
1~0~2~3*
```

Formatted String Literals (f-strings) with Print Function

Syntax

```
print(f"<string> {<expression>}")
```

- `<string>`: A string literal that may contain placeholders.
- `<expression>`: Any valid Python expression that will be evaluated and formatted into the string.

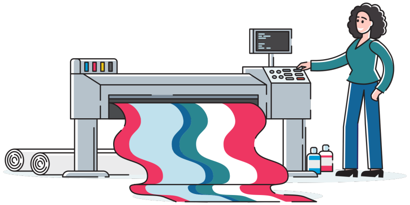
Output formatting can be done directly within the f-string using curly braces {} to include variables or expressions.

Output Data: Use of f-strings

```python
# Filename: use_of_fstrings.py

def main():
    # Use a semicolon to write multiple statements in a single line
    x = "A+"; y = 'A'

    print(f"Desmond will get {x} and Sunny will get {y}")
    print(f"No! Desmond will get {y} and Sunny will get {x}")

if __name__ == "__main__":
    main()
```

Output:

```
Desmond will get A+ and Sunny will get A
No! Desmond will get A and Sunny will get A+
```



Output Data: Printing Different Types of Data

```python
# Filename: printing_different_types.py

def main():
    name = "Desmond"
    gender = 'M'
    age = 18
    height = 1.72
    print("Hi", name)
    print("Age:", age)
    print("Gender", gender)
    print("Height", height, 'm')

if __name__ == "__main__":
    main()
```

Output:

```
Hi Desmond
Age: 18
Gender M
Height 1.72 m
```

Output Data: Printing Multiline Texts

```python
# Filename: printing_multiline.py

def main():
    print('''COMP 1023 is a Python Course.
I think it is the best COMP course!
''')

if __name__ == "__main__":
    main()
```

Output:

```
COMP 1023 is a Python Course.
I think it is the best COMP course!
```

- We can print a paragraph using the print function by enclosing the text in triple quotes (i.e., `'''` or `"""`).
- It allows us to include line breaks and preserve formatting.

Note

- In Python, triple quotes (`'''` or `"""`) can be used for:
  - Multiline strings
  - Docstrings (which document functions, classes, or modules; more details on this later)
  - Comments

You can use triple quotes for multiline strings and documentation, but for comments, it is better to use the # symbol.

Output Data: Printing a Long Line of Texts

```python
# Filename: printing_long_line.py

def main():
    print("COMP 1023 is a Python Course. \
I think it is the best COMP course!")

if __name__ == "__main__":
    main()
```

Output:

```
COMP 1023 is a Python Course. I think it is the best COMP course!
```

- The backslash can be used to indicate that a statement continues on the next line.
- This is useful for long lines of code that you want to break for readability.

Getting User's Input from the Standard Input Device

- We can take input from the user using the input() function.

Syntax

```
input(<prompt>)
```

- Parameter:
  - `<prompt>`: The string we wish to display on the screen. It is optional.

Note: The entered data is a string.

```
age = input("Enter your age: ")
print(age)                 # Assume the input is 18. It prints 18
print(type(age))           # Print <class 'str'>
print(type(int(age)))      # Print <class 'int'>
print(type(float(age)))    # Print <class 'float'>
age = int(age) + 1         # Convert age to int and increase it by 1
print("Now you are", age, "years old") # Print Now you are 19 years old
```

Indentation

- The difference in indentation DOES DISTURB Python's ability to translate the source code into bytecode.

```python
print("Welcome to COMP 1023")
  print("Poor indentation")
```

Error:

```
    print("Poor indentation")
    ^
IndentationError: unexpected indent
```

Good Programming Style

- Put no more than one statement on each line.
  (a semicolon is needed to separate two statements.)
  `print("COMP 1023 is good"); print("It is the best COMP course")`
- Indent statements properly.
- Use blank lines to separate parts of the program.
- Include comments for clarity.
- Use type hinting for better code readability.
- Define a main function for structured code execution.

Key Terms

- Boolean
- Case-sensitive language
- Comments
- Data types
- Floating point
- Indentation
- Identifiers
- Immutable
- Initialization
- Integer
- Literals
- Mutable
- Named constants
- Objects
- Reserved words
- Snake case
- String
- Type conversion
- Type hinting
- Variables

Review Questions

Fill in the blanks in each of the following sentences about the Python environment.

1. ____________________ in programs are for human to read, and to increase their readability. They are added using ____________________ and ____________________.
2. ____________________ is the entry point of a Python program/script.
3. ____________________ are used to store data in a program. They are created by being assigned a ____________________, and should be referenced using a ____________________.
4. An ____________________ is the name used for an element in a program. It is a sequence of characters of any length consists of ____________________, ____________________, and ____________________. It must start with a ____________________ or an ____________________. It cannot start with a ____________________. It cannot be a ____________________.
5. ____________________ are values associated with data types.

Answer: 1. Comments; hash symbols; triple quotes, 2. main(), 3. objects; value; variable, 4. identifier; letters; digits; underscores; letter; underscore; digit; reserved word, 5. Literals.

Review Questions

Fill in the blanks in each of the following sentences about the Python environment.

6. It is a good practice to specify the type of a variable using a ________________. We call it ________________.
7. A ________________ represents permanent data that never changes, and its variable uses all ________________ letters.
8. You can output data to the standard output device using the ________________ function. The parameter ________________ is used to specify the separator, and ________________ is used to specify the ending character.
9. You can get input using the ________________ function and convert a string into a numerical value using the ________________ function or the ________________ function.
10. In Python, variable names with more than one word are separated using ________________, and we call it ________________.
11. The difference in ________________ disturbs Python's ability to translate the source code.

Answer: 6. colon, type hinting, 7. named constant, capital, 8. print, sep, end, 9. input, int, float, 10. underscores, snake case, 11. indentation.

Further Reading

- Read Sections 2.1 - 2.8 and 2.12 of “Introduction to Python Programming and Data Structures” textbook.

That's all!

Any questions?