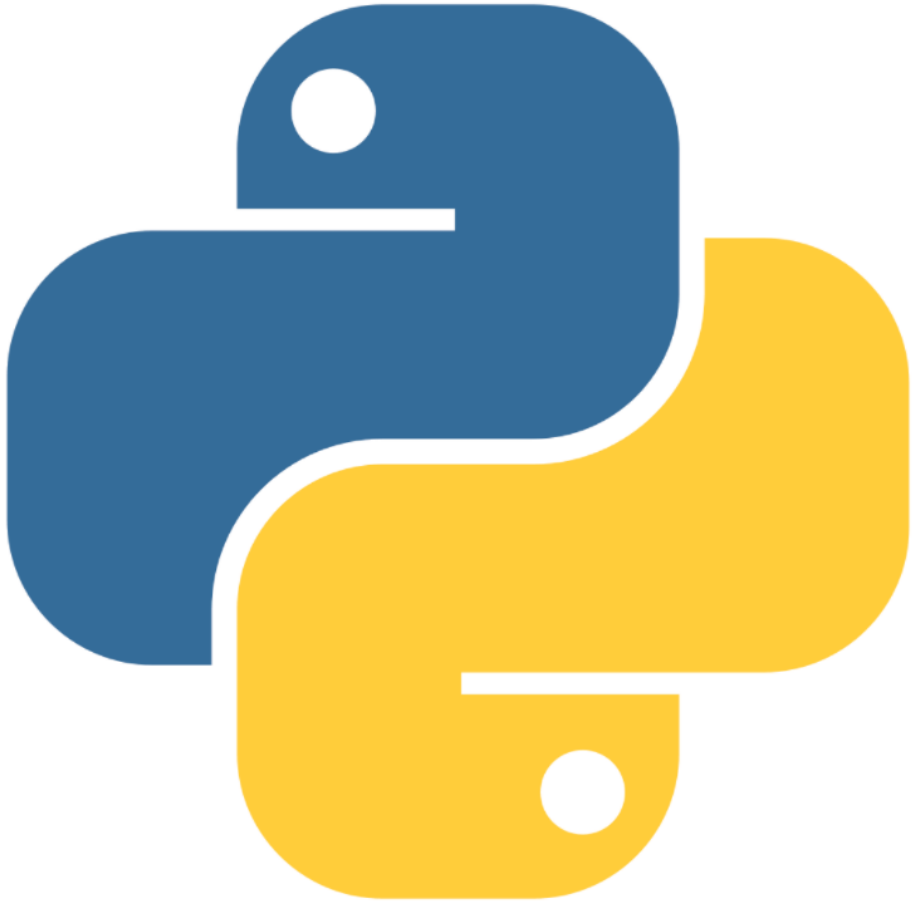
PYTHON



Hoang Oanh Pham

Content

- 1.Introduction
- 2.Variable
- 3.Arithmetic operator
- 4.Input, output
- 5.Error
- 6.List, set, dictionary
- 7.Control flow: if-else, for loop, while loop
- 8. Class
- 9.Regular expression

Why Python?

- Python is
  - Simple, easy to learn syntax
  - Powerful and flexible
  - Short and easy to read code
  - Large Collection of additional Libraries for special tasks

Source Code

- **Code** is the plain text representation of the program.
- A **line** of code is a single row of text.
- A **statement** is an instruction in the code.
- A **program** is just a collection of statements executed in order.

Variables

- References to locations in memory.
- Created by using the assignment operation using the = symbol (assignment operator)
- The name of the variable is on the left side of the “equals” sign
- The value of the variable is on the right side of the “equals” sign
- Ex:
  - a=5
  - b=9

a	5
b	9

Variable Naming Rules

- Letters (a – z or A – Z) , digits (0 – 9), or underscore (_) characters.
- The first character must be a letter or an underscore
- The name cannot contain spaces or any other special characters or reserved word (keyword)
- Always choose a meaningful name
- Python is case sensitive
- Upper and lower case letters are not the same ($A \neq a$)

Variables Type:

- Tells us what kind of data the variable holds
- The same operators act differently on different types
  - a = 7
  - b = ‘cat’
  - c=2
  - print(a*b) #’catcatcatcatcatcatcat’
  - print(a*c) #14
- Multiplication means duplicate for strings
- Types:
  - Number: Integer(whole numbers), Float(decimal numbers)
  - String

Arithmetic Operators

SYMBOL	OPERATION	Description/Notes
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another Also use for negation
*	Multiplication	Multiplies two numbers
/	Division	Divides left hand operand by right hand operand
//	Floor Division	Rounds down the result of a division to the closest whole number value.
%	Remainder, or modulo	Divides left hand operand by right hand operand and returns remainder
**	Exponent	Performs exponential (power) calculation on operators

Combined Operators:

Operator	Example Usage	Equivalence
+=	x += 5	x = x + 5
-=	y -= 2	y = y -2
*=	z *= 10	z = z * 10
/=	a /= b	a = a / b
%=	c %= 3	c = c % 3

Input, Output

- To get input from the user we use the **`input()`** built-in function
- The **`print`** function can be used to display the value of a variable
- The input function always returns a string type (text)

Ex:

```
>>> print ("Enter your name: ")
Enter your name:
>>> name = input()
```

Errors:

- Syntax errors
- Logic errors
- Run-time errors

Syntax Errors

- Syntax error is a mistake such as a:
  - Misspelled word
  - Missing punctuation character
  - Incorrect use of an operator
- A syntax error occurs when a statement in the program violates the rules of the programming language
- A syntax error must be fixed before the program can be executed
- The interpreter will generate a message when encountering a syntax error.

Logic errors

- A logic error causes the program to operate incorrectly, but not to fail.
- The interpreter does not find these errors

Run-time errors

- Common examples:
  - dividing by zero
  - referencing missing files
  - calling invalid functions
  - not handling certain input correctly

List

- Lists contains a collection or sequence of values
- Python can create lists of any type
- Lists can contain strings, numbers, even other lists
- List can contain a mix of types
- Each item in the list is called an element
- We can use lists to process a variety of types of data.
- To define a list, use the `[]` and separate the elements with commas.

Access to elements

- We use the subscript operator [] to access elements in a list
- Use a valid index value
  - An integer value
  - First element is at index zero
  - Index of the last element is the number of elements minus 1
- Negative numbers can be used to access elements from the rightmost element of the list
  - Use the colon [:] to get slices of a list

sample.py <untitled> *

```
1 myList = ["apple", "banana", "cherry", "orange", "pear", "cucumber", "mango"]
2 print(myList[-1])
3 print(myList[3:6])
```

```
>>> %Run sample.py
mango
['orange', 'pear', 'cucumber']
```

Methods

- **`append(element)`** adds `element` at the end of the list.
- **`remove(element)`** removes the first occurrence of `element` from the list, if it's there.
- **`pop(index)`** removes the element at the given `index`.
- **`index(element)`** finds the index of the first occurrence of `element` in the list.
- **`count(element)`** tells you the number of times that `element` appears in the list. It returns an integer.

Dictionary:

- A container used to describe associative relationships
- Represented by the **dict** object type
- A dictionary maps keys with values
  - Key is a term that can be located in the dictionary
  - Keys are unique- each one can only be used once
  - Could be: string, tuple, or number
  - Value describe data associated with key
  - Any type
- To define a dictionary, use the **{}** to surround **key:value** pairs.
- Separate **key:value** pairs with commas

Access to elements

- Use the key inside the []
- Entries in a dictionary can be added, deleted and modified as needed
  - **dictionary[key] = value** adds a new pair if it doesn't exist
  - **dictionary[key] = value** modifies existing entry if it exists
  - **del dictionary[key]** deletes entry if it exists

sample.py <untitled> *

```
1 thisdict = {
2   "brand": "Ford",
3   "model": "Mustang",
4   "year": 1964
5 }
6 print(thisdict["brand"])
7 del thisdict["brand"]
8 print(thisdict)
```

```
>>> %Run sample.py
Ford
{'model': 'Mustang', 'year': 1964}
```

Set

- unordered collection of unique elements
- Elements do not have a position or index.
- Elements are unique: No elements in the set share the same value.
- A set can be created using the `set()` function, which accepts a sequence-type iterable object (list, tuple, string, etc.)
- A set literal can be written using curly braces { } with commas separating set elements.
- Note that an empty set can only be created using set `()`

Operations:

- `len(set1)` Number of Elements in Set
- `set1.update(set2)` Add all elements from `set2` into `set1`
- `set.add(value)` Add `value` to set
- `set.remove(value)` Remove `value` from set
- `set.pop()` Remove an arbitrary element from set
- `set.clear()` Clears all elements from set

If-else:

- if expression:
  - Statement(s)
- else:
  - Statement(s)

Ex

sample.py

```
1  import os
2  import sys
3
4  a = 33
5  b = 200
6  if b > a:
7      print("b is greater than a")
8  else:
9      print("b is less than a")
```

```
>>> %Run sample.py
b is greater than a
```

For loop

- for val in sequence:

  Loop body

sample.py <untitled> *

```
1 my_dict = {'color': 'blue', 'fruit': 'apple', 'pet': 'dog'}
2 for key in my_dict:
3     print(key, '->', my_dict[key])
4
```

```
>>> %Run sample.py
color -> blue
fruit -> apple
pet -> dog
```

Ex

sample.py

```
1 listA = [1,2,3,4,5]
2 for x in listA:
3     print(x)
```

```
>>> %Run sample.py
1
2
3
4
5
```

While loop

- While test_expression:
  - Body of while

Ex

sample.py sample.py

```
1 myList = ['pineapple', 'banana', 'watermelon', 'mango']
2
3 index = 0
4 while index < len(myList):
5     element = myList[index]
6     print(len(element))
7     index += 1
```

```
>>> %Run sample.py
 9
 6
 10
 5
```

functions

- Function: a named piece of code that performs a specific task
- Function call: invoking the function name causes the function to execute.
- Arguments and Parameters: values given as input to the function.

- def functionname(parameter):
  Function code
  return [expression]

Ex:

sample.py *

```
1  def square(x):
2      return x*x
3  print(square(5))
```

```
>>> %Run sample.py
25
```

Class

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.

Ex

sample.py

```python
class Person:
    def __init__(self, age, name):
        self.age = age
        self.name = name
    def greet(self):
        print(f'Hello, {self.name}. You are {self.age} years old.')

person1 = Person(age=20, name="Rose")
person1.greet()
```

```
>>> %Run sample.py
Hello, Rose. You are 20 years old.
```

Regular expression:

- A regular expression is a special string (a sequence of characters)
- Describes a search pattern, each regular expression matches a set of strings

  Each dot . and ? must match exactly one character

  […] matches any listed character

  * matches anything including an empty string

  ^$ beginning and end of line

Ex:

```
ls a?.txt

a1.txt a2.txt ab.txt
```

```
ls lab1.*

lab1. lab1.c lab1.doc

lab1.docx lab1.pdf
```

```
ls lab1.???

lab1.doc lab1.pdf
```

```
ls a[ab]*.???

abcd.txt abc.txt ab.txt
```

Sources:

- https://www.programiz.com/python-programming/class
- Computer Programming Courses – Drexel University

Question?

Thank you!!!