



Basic Python Programming

Peerapon Vateekul, Ph.D.

peerapon.v@4amconsult.com



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words

Basic Python

- Python is a high-level, dynamically typed multiparadigm programming language.
- Python code is often said to be almost like pseudocode, since it allows you to express very powerful ideas in very few lines of code while being very readable.
- As an example, here is an implementation of the classic quicksort algorithm in Python:

```
def quicksort(arr):  
    if len(arr) <= 1:  
        return arr  
    pivot = arr[len(arr) // 2]  
    left = [x for x in arr if x < pivot]  
    middle = [x for x in arr if x == pivot]  
    right = [x for x in arr if x > pivot]  
    return quicksort(left) + middle + quicksort(right)  
  
print(quicksort([3,6,8,10,1,2,1]))  
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```



Basic Python

- There are currently two different supported versions of Python, 2 and 3
- Python 3 introduced many backwards-incompatible changes to the language
 - code written for 2 may not work under 3 and vice versa.
 - **For this class all code will use Python 3.**
- You can check your Python version at the command line by running
 - `python --version`.

```
1 !python --version
```

```
Python 3.9.16
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words



Python Data Types

■ Basic/Primitive Data Types

- Number
 - Integer, Float
- Boolean
- String (Object)

■ Collection Data Types

- List
- Tuple
- Dictionary
- Set

■ Numpy (Next Section)

■ Pandas and Series (Next Section)



Python Data Types

- Basic/Primitive Data Types

■ Basic/Primitive Data Types

- Number
 - Integer, Float
- Boolean
- String (Object)

Like most languages, Python has a number of basic types including integer, float, boolean, and string.



Python Data Types

- Basic/Primitive Data Types (Number)

- **Number**: Integer and float work as you would expect from other languages:
 - Note that unlike many languages, Python **does not** have unary increment (x++) or decrement (x--) operators.

3

Integer

3.14

Float

Operation	Result
$x + y$	sum of x and y
$x - y$	difference of x and y
$x * y$	product of x and y
x / y	quotient of x and y
$x // y$	floored quotient of x and y
$x \% y$	remainder of x / y
$x ** y$	x to the power y
<code>abs(x)</code>	absolute value or magnitude of x
<code>int(x)</code>	x converted to integer
<code>float(x)</code>	x converted to floating point

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)       # Prints "4"
x *= 2
print(x)       # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

Reference:

<http://cs231n.github.io/python-numpy-tutorial/>
<https://docs.python.org/3/library/stdtypes.html>



Python Data Types

- Basic/Primitive Data Types (Number)

```
>>> 7+3
10
>>> 7-3
4
>>> 7*3
21
>>> 7/3
2.3333333333333335
>>> 7//3
2
>>> 7%3
1
>>> 2**10
1024
>>> 1.44**0.5
1.2
```

+	บวก
-	ลบ
*	คูณ
/	หาร
//	หารปัดเศษ
%	เศษจากการหาร
**	ยกกำลัง



Python Data Types

- Basic/Primitive Data Types (Number)

```
>>> r = 0
>>> r = r + 100
>>> r = r - 30
>>> r = r / 2
>>> r = r // 3
>>> r = r % 7
>>> r = r ** 2
>>> print(r)
```

.....

```
>>> r = 0
>>> r += 100
>>> r -= 30
>>> r /= 2
>>> r //= 3
>>> r %= 7
>>> r **= 2
>>> print(r)
```

.....

แบบไหนเข้าใจง่ายกว่า ?



Python Data Types

- Basic/Primitive Data Types (Boolean)

- **Boolean:** Python implements all of the usual operators for Boolean logic, but uses English words rather than symbols (&&, ||, etc.):

- Boolean Operations

- and
- or
- not
- !=
- ==

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```



Python Data Types

- Basic/Primitive Data Types (String)

- String: Textual data type

```
hello = 'hello'      # String literals can use single quotes
world = "world"      # or double quotes; it does not matter.
print(hello)         # Prints "hello"
print(len(hello))    # String length; prints "5"
hw = hello + ' ' + world # String concatenation
print(hw)            # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12) # sprintf style string formatting
print(hw12)          # prints "hello world 12"
```



Python Data Types

- Basic/Primitive Data Types (String)

- String objects have a bunch of useful methods; for example:

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints "  hello  "
print(s.replace('l', '(ell)')) # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world  '.strip()) # Strip leading and trailing whitespace; prints "world"
```

You can find a list of all string methods [in the documentation](https://docs.python.org/3.5/library/stdtypes.html#string-methods).
(<https://docs.python.org/3.5/library/stdtypes.html#string-methods>)

+ Python Data Types

- Basic/Primitive Data Types (Type Conversion)

```
>>> i = 3
>>> f = 3.0
>>> s = "3"
>>> t = "3.0"
>>> i += int(f)      # เปลี่ยน float เป็น int
>>> i += int(s)      # เปลี่ยนสตริงเป็น int
>>> f += float(t)    # เปลี่ยนสตริงเป็น float
>>> s += str(i)      # เปลี่ยน int เป็นสตริง
>>> s = s + str(f)   # เปลี่ยน float เป็นสตริง
>>> print(s)
```



Python Data Types

- Collection Data Types

■ Collection Data Types

- List
- Tuple
- Dictionary
- Set

+ Python Data Types

- Collection Data Types (List)

■ List

- A list is the Python equivalent of an array, but is resizeable and can contain elements of different types:

3	1	2
---	---	---

3	1	foo
---	---	-----

3	1	foo	bar
---	---	-----	-----

3	1	foo
---	---	-----

```
xs = [3, 1, 2]      # Create a list
print(xs, xs[2])   # Prints "[3, 1, 2] 2"
print(xs[-1])      # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'      # Lists can contain elements of different types
print(xs)          # Prints "[3, 1, 'foo']"
xs.append('bar')    # Add a new element to the end of the list
print(xs)          # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()        # Remove and return the last element of the list
print(x, xs)        # Prints "bar [3, 1, 'foo']"
```

+ Python Data Types

- Collection Data Types (List)

- **Slicing:** In addition to accessing list elements one at a time, Python provides concise syntax to access sublists; this is known as *slicing*:

```
nums = list(range(5))      # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])            # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]"
```



Python Data Types

- Collection Data Types (Tuple)

■ Tuple

- A tuple is an (**immutable**) ordered list of values.
- A tuple is in many ways similar to a list; one of the most important differences is that tuples can be used as keys in dictionaries and as elements of sets, while lists cannot. Here is a trivial example:

```
d = {(x, x + 1): x for x in range(10)} # Create a dictionary with tuple keys
t = (5, 6)                            # Create a tuple
print(type(t))                        # Prints "<class 'tuple'>"
print(d[t])                           # Prints "5"
print(d[(1, 2)])                      # Prints "1"
```



Python Data Types

- Collection Data Types (Dictionary)

■ Dictionary

- A dictionary stores (key, value) pairs, similar to a [Map](#) in Java or an object in Javascript. You can use it like this

```
d = {'cat': 'cute', 'dog': 'furry'} # Create a new dictionary with some data
print(d['cat'])                    # Get an entry from a dictionary; prints "cute"
print('cat' in d)                  # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet'                  # Set an entry in a dictionary
print(d['fish'])                    # Prints "wet"
# print(d['monkey'])               # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A'))      # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))        # Get an element with a default; prints "wet"
del d['fish']                      # Remove an element from a dictionary
print(d.get('fish', 'N/A'))        # "fish" is no longer a key; prints "N/A"
```

+ Python Data Types

- Collection Data Types (Set)

■ Sets

- A set is an unordered collection of **distinct** elements. As a simple example, consider the following:

```
animals = {'cat', 'dog'}  
print('cat' in animals)    # Check if an element is in a set; prints "True"  
print('fish' in animals)   # prints "False"  
animals.add('fish')        # Add an element to a set  
print('fish' in animals)   # Prints "True"  
print(len(animals))        # Number of elements in a set; prints "3"  
animals.add('cat')         # Adding an element that is already in the set does nothing  
print(len(animals))        # Prints "3"  
animals.remove('cat')      # Remove an element from a set  
print(len(animals))        # Prints "2"
```


+ Python Data Types

- Collection Data Types

	list	tuple	dict	set
การใช้	- ลำดับของข้อมูลมีความหมาย อาจมีการเปลี่ยนแปลง - ข้อมูลในรายการมักมีความหมายเดียวกัน	- ลำดับของข้อมูลมีความหมาย - สร้างแล้วไม่เปลี่ยนแปลง - ข้อมูลใน tuple มักมีความหมายต่างกัน	- เก็บข้อมูลเป็นคู่ๆ key-value โดยใช้ key เข้าถึงข้อมูลเพื่อให้ได้ value มาใช้งาน	- เก็บข้อมูลไม่ซ้ำ ลำดับของข้อมูลไม่มีความหมาย เพื่อตรวจสอบว่า มีข้อมูลหรือไม่ รองรับ set operations
การเข้าใช้ข้อมูล	ใช้จำนวนเต็มระบุตำแหน่ง <code>d[i]</code>	ใช้จำนวนเต็มระบุ <code>d[i]</code>	ใช้ key เป็นตัวระบุตำแหน่งข้อมูล <code>d[key]</code>	ต้อง <code>for...in...</code> เพื่อแจ้งข้อมูล
การค้นด้วย in	ค้นจากซ้ายไปขวา	ค้นจากซ้ายไปขวา	มีวิธีค้นที่เร็วมาก	มีวิธีค้นที่เร็วมาก
การสร้าง	<code>x = [1,2,3,4]</code>	<code>t = (1,2,3,4)</code>	<code>d = { "k1":1, "k2":2 }</code>	<code>s = {1,2,3,4}</code>
การเพิ่มข้อมูล	<code>x.append(3)</code> <code>x.insert(1,99)</code>	สร้างแล้วเปลี่ยนแปลงไม่ได้ ต้องสร้างใหม่ <code>t = t + (4,)</code>	<code>d["k1"] = 1</code> <code>d["k2"] = 2</code> หรือใช้ <code>update</code>	<code>s.add(3)</code>



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words

Condition

■ What are if...else statement in Python?

- Decision making is required when we want to execute a code only if a certain condition is satisfied.
- The if...elif...else statement is used in Python for decision making.

■ Python if Statement Syntax

```
if test expression:  
    statement(s)
```

- Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.
- If the text expression is False, the statement(s) is not executed.
- In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end.
- Python interprets non-zero values as True. None and 0 are interpreted as False.

■ Python if Statement Flowchart

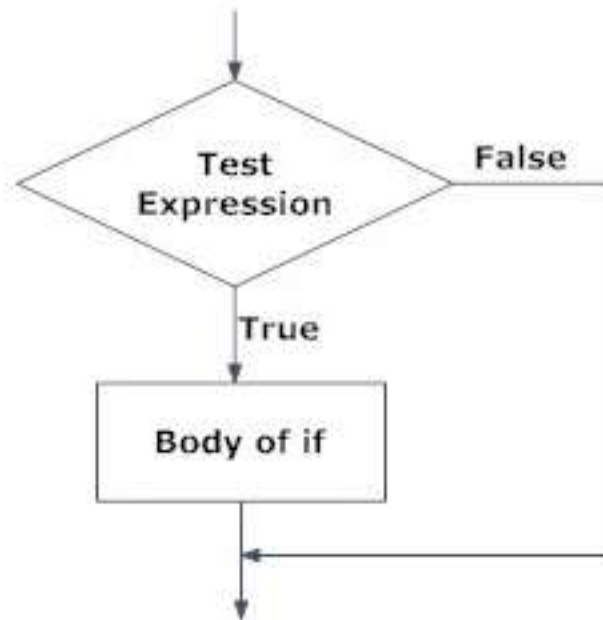


Fig: Operation of if statement

■ Example: Python if Statement

```
script.py  IPython Shell
1  # If the number is positive, we print an appropriate message
2
3  num = 3
4  if num > 0:
5      print(num, "is a positive number.")
6  print("This is always printed.")
7
8  num = -1
9  if num > 0:
10     print(num, "is a positive number.")
11 print("This is also always printed.")
```

OUTPUT

3 is a positive number.
This is always printed.
This is also always printed.



Condition

■ Python if...else Statement

■ Syntax of if...else

```
if test expression:  
    Body of if  
else:  
    Body of else
```

- The if..else statement evaluates test expression and will execute body of if only when test condition is True.
- If the condition is False, body of else is executed. Indentation is used to separate the blocks.

Condition

■ Python if...else Statement

■ Syntax of if...else

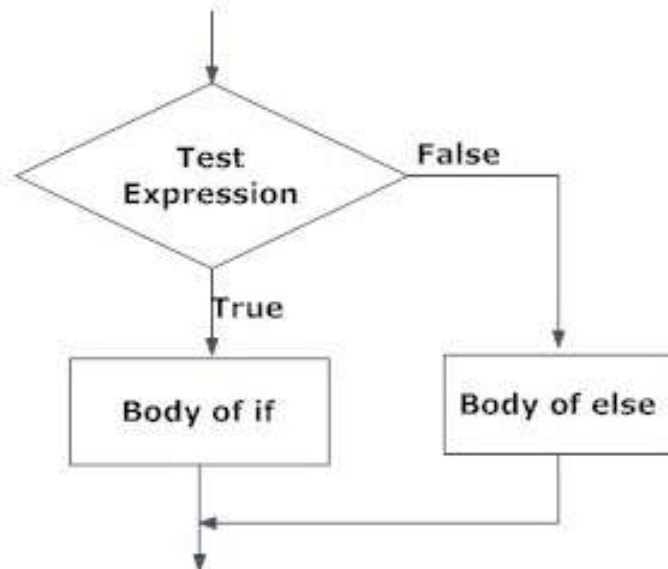


Fig: Operation of if...else statement

■ Example: Python if Statement

```
script.py  IPython Shell
1  # In this program,
2  # we check if the number is positive or
3  # negative or zero and
4  # display an appropriate message
5
6  num = 3.4
7
8  # Try these two variations as well:
9  # num = 0
10 # num = -4.5
11
12 if num > 0:
13     print("Positive number")
14 elif num == 0:
15     print("Zero")
16 else:
17     print("Negative number")
```

OUTPUT

Positive
Number

Condition

■ Python Nested if statements

- We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming.
- Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

```
# In this program, we input a number
# check if the number is positive or
# negative or zero and display
# an appropriate message
# This time we use nested if

num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

OUTPUT

Output 1

```
Enter a number: 5
Positive number
```

Output 2

```
Enter a number: -1
Negative number
```

Output 3

```
Enter a number: 0
Zero
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words



Loop

■ What is for loop in Python?

- The for loop in Python is used to iterate over a sequence ([list](#), [tuple](#), [string](#)) or other iterable objects. Iterating over a sequence is called traversal.

■ Syntax of for Loop

```
for val in sequence:  
    Body of for
```

- Here, val is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

+ Loop

31

■ Flowchart of for Loop

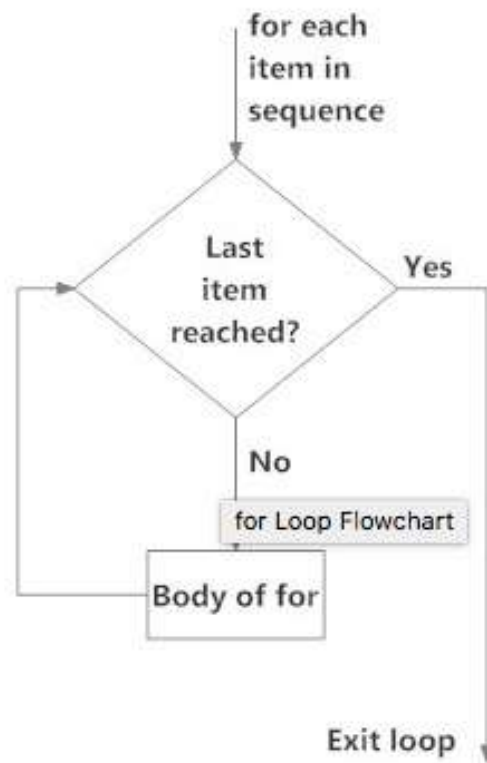


Fig: operation of for loop

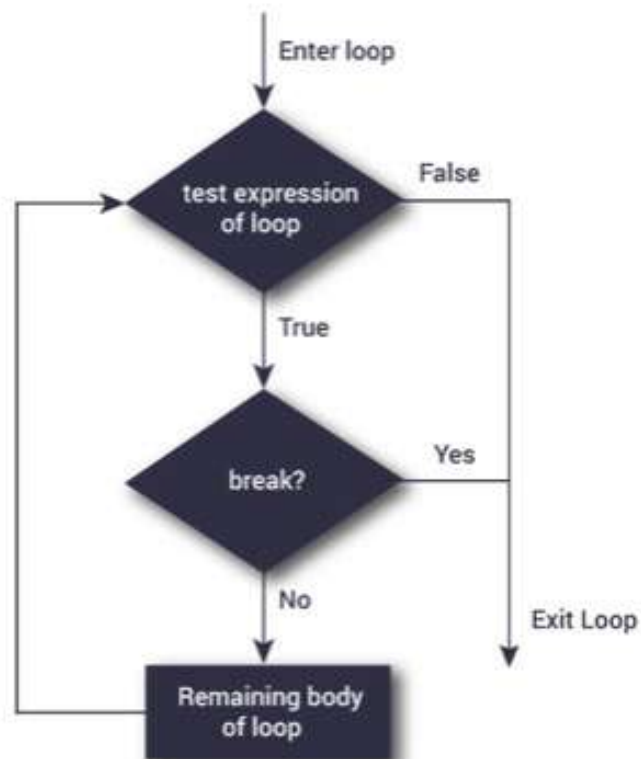
■ Example: Python for Loop

```
script.py  IPython Shell
1  # Program to find the sum of all numbers stored in a list
2
3  # List of numbers
4  numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
5
6  # variable to store the sum
7  sum = 0
8
9  # iterate over the list
10 for val in numbers:
11     sum = sum+val
12
13 # Output: The sum is 48
14 print("The sum is", sum)
```

The sum is 48

+ Loop - Break

■ Flowchart of for Loop



■ Example: Python for Loop

```
script.py  IPython Shell
1  # Use of break statement inside loop
2
3  for val in "string":
4      if val == "i":
5          break
6      print(val)
7
8  print("The end")
```

s
t
r
The end



Loop

- Continue

■ Python continue statement

- The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

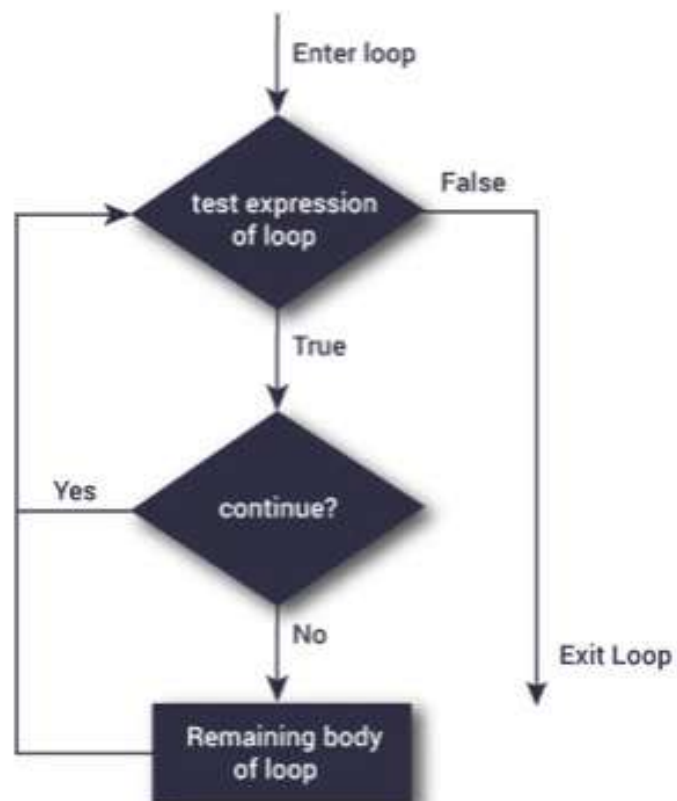
■ Syntax of Continue

```
continue
```

+ Loop

- Continue

■ Flowchart of for Loop



■ Example: Python for Loop

```
script.py  IPython Shell
1  # Program to show the use of continue statement inside loops.
2
3  for val in "string":
4      if val == "i":
5          continue
6          print(val)
7
8  print("The end")
```

s
t
r
i
n
g
The end



Loop - List

- **Loops:** You can loop over the elements of a list like this:

```
animals = ['cat', 'dog', 'monkey']  
for animal in animals:  
    print(animal)  
# Prints "cat", "dog", "monkey", each on its own line.
```

+ Loop

- List Comprehension

- **List comprehensions:** When programming, frequently we want to transform one type of data into another. As a simple example, consider the following code that computes square numbers:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)    # Prints [0, 1, 4, 9, 16]
```

- You can make this code simpler using a **list comprehension**:

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)    # Prints [0, 1, 4, 9, 16]
```

+ Loop

- List Comprehension with If

- List comprehensions: (cont.)
- List comprehensions can also contain conditions:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Loop

- Range

■ The range() function

- We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers).
- We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.
- This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.
- To force this function to output all the items, we can use the function list().

```
# Output: range(0, 10)
print(range(10))
```

```
# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))
```

```
# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))
```

```
# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```



Loop

- Range with len

- We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate through a sequence using indexing. Here is an example.

```
script.py  IPython Shell
1  # Program to iterate through a list using indexing
2
3  genre = ['pop', 'rock', 'jazz']
4
5  # iterate over the list using index
6  for i in range(len(genre)):
7      print("I like", genre[i])
```

OUTPUT

I like pop
I like rock
I like jazz

+ Loop - While

- **What is while loop in Python?**
- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- We generally use this loop when we don't know beforehand, the number of times to iterate.
- **Syntax of while Loop in Python**

```
while test_expression:  
    Body of while
```


Loop - While

■ Flowchart of for Loop

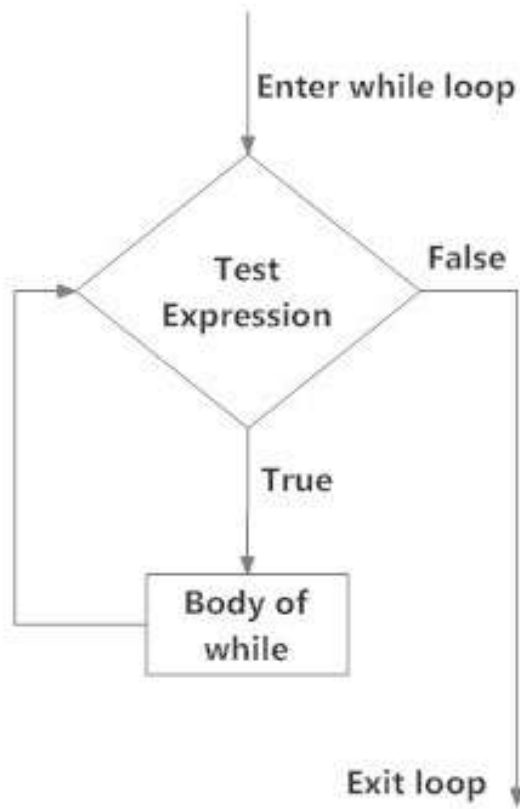


Fig: operation of while loop

■ Example: Python for Loop

```
script.py  IPython Shell
1  # Program to add natural
2  # numbers upto
3  # sum = 1+2+3+...+n
4
5  # To take input from the user,
6  # n = int(input("Enter n: "))
7
8  n = 10
9
10 # initialize sum and counter
11 sum = 0
12 i = 1
13
14 while i <= n:
15     sum = sum + i
16     i = i+1    # update counter
17
18 # print the sum
19 print("The sum is", sum)
```

Enter n: 10
The sum is 55



Loop - While

■ What is the use of break and continue in Python?

- In Python, break and continue statements can alter the flow of a normal loop.
- Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
- The break and continue statements are used in these cases.

■ Python break statement

- The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words

Functions

- Python functions are defined using the **def** keyword. For example:

```
def sign(x):  
    if x > 0:  
        return 'positive'  
    elif x < 0:  
        return 'negative'  
    else:  
        return 'zero'  
  
for x in [-1, 0, 1]:  
    print(sign(x))  
# Prints "negative", "zero", "positive"
```




Functions

- Parameter

- We will often define functions to take optional keyword arguments, like this:

Parameter Default Parameter



```
def hello(name, loud=False):  
    if loud:  
        print('HELLO, %s!' % name.upper())  
    else:  
        print('Hello, %s' % name)  
  
hello('Bob') # Prints "Hello, Bob"  
hello('Fred', loud=True) # Prints "HELLO, FRED!"
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words



Lambda Function

- **What are lambda functions in Python?**
- In Python, anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.

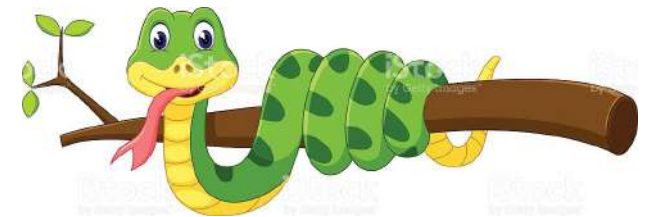


Lambda Function

- **How to use lambda Functions in Python?**
- A lambda function in python has the following syntax.
- **Syntax of Lambda Function in python**

```
lambda arguments: expression
```

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.





Lambda Function

- In the above program, `lambda x: x * 2` is the lambda function. Here `x` is the argument and `x * 2` is the expression that gets evaluated and returned.
- This function has no name. It returns a function object which is assigned to the identifier `double`. We can now call it as a normal function. The statement

```
double = lambda x: x * 2
```

- is nearly the same as

```
def double(x):  
    return x * 2
```



Lambda Function

50

- Use of Lambda Function in python
- We use lambda functions when we require a nameless function for a short period of time.
- In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like **filter()**, **map()** etc.

+ Lambda Function

- Filter

- Example use with `filter()`
- The `filter()` function in Python takes in a function and a list as arguments.
- The function is called with all the items in the list and a new list is returned which contains items for which the function evaluates to True.
- Here is an example use of `filter()` function to filter out only even numbers from a list.

```
script.py  IPython Shell
1  # Program to filter out only the even items from a list
2
3  my_list = [1, 5, 4, 6, 8, 11, 3, 12]
4
5  new_list = list(filter(lambda x: (x%2 == 0) , my_list))
6
7  # Output: [4, 6, 8, 12]
8  print(new_list)
```

+ Lambda Function

- Map

- **Example use with map()**
- The **map()** function in Python takes in a function and a list.
- The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.
- Here is an example use of map() function to double all the items in a list.

```
script.py  IPython Shell
1  # Program to double each item in a list using map()
2
3  my_list = [1, 5, 4, 6, 8, 11, 3, 12]
4
5  new_list = list(map(lambda x: x * 2 , my_list))
6
7  # Output: [2, 10, 8, 12, 16, 22, 6, 24]
8  print(new_list)
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words

- The syntax for defining classes in Python is straightforward:

```
class Greeter(object):  
  
    # Constructor  
    def __init__(self, name):  
        self.name = name # Create an instance variable  
  
    # Instance method  
    def greet(self, loud=False):  
        if loud:  
            print('HELLO, %s!' % self.name.upper())  
        else:  
            print('Hello, %s' % self.name)  
  
g = Greeter('Fred') # Construct an instance of the Greeter class  
g.greet() # Call an instance method; prints "Hello, Fred"  
g.greet(loud=True) # Call an instance method; prints "HELLO, FRED!"
```

+ Special class functions

```
class Rational:
    def __init__(self, n, d):
        g = Rational.gcd(n, d)
        self.nu = n//g # numerator เศษ
        self.de = d//g # denominator ส่วน

    def add(self, x):
        ...
    def mult(self, x):
        ...
    def to_float(self):
        ...
    def less_than(self, x):
        ...
    def to_str(self):
        ...
```

```
r3 = r1.add(r2)
print(to_str(r3), to_float(r3) )
r4 = r1.mult(r2)
print( r3.less_than(r4) )
```

```
class Rational:
    def __init__(self, n, d):
        g = Rational.gcd(n, d)
        self.nu = n//g # numerator เศษ
        self.de = d//g # denominator ส่วน

    def __add__(self, x):
        ...
    def __mul__(self, x):
        ...
    def __float__(self):
        ...
    def __lt__(self, x):
        ...
    def __str__(self):
        ...
```

```
r3 = r1 + r2
print( str(r3), float(r3) )
r4 = r1 * r2
print( r3 < r4 )
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words

Python Packages

- Built-in Packages

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



Python Packages

- Import Other Packages

- The difference between import and from import in Python is:
 - **import** imports an entire code library.
 - **from import** imports a specific member or members of the library.

- Example:

- **import** [lib_name]
- **import** [lib_name] **as** [variable_name]
- **import** [lib_name].[object_name]
- **from** [lib_name] **import** [object_name1]
- **from** [lib_name] **import** [object_name1], [object_name2]
- **from** [lib_name] **import** *

```
import datetime

now = datetime.datetime.now()
print(now)
```

2020-06-04 18:06:19.230703

```
import datetime as dt

now = dt.datetime.now()
print(now)
```

2020-06-04 18:06:19.239702

datetime

- datetime
 - now
- timedelta
- etc.

```
from datetime import datetime

now = datetime.now()
print(now)
```

2020-06-04 18:06:19.248703

```
from datetime import datetime, timedelta

now = datetime.now()
print(now)
```

2020-06-04 18:06:19.264702

```
from datetime import *

now = datetime.now()
print(now)
```

2020-06-04 18:06:19.282703



Python Packages

- Install Python Packages

■ Linux Command

■ Install package

- `pip install <package name>`
- `pip install <package name> --upgrade`

■ Check installed package

- `pip freeze`

Command Line

```
PS C:\Windows\system32> pip install pandas --upgrade
Collecting pandas
  Downloading pandas-1.0.3-cp37-cp37m-win_amd64.whl (8.7 MB)
    | 8.7 MB 939 kB/s
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2019.3)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in c:\programdata\anaconda3\lib\site-packages (from pandas) (1.16.5)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in c:\programdata\anaconda3\lib\site-packages (from pandas) (2.8.0)
Requirement already satisfied, skipping upgrade: six>=1.5 in c:\programdata\anaconda3\lib\site-packages (from python-dateutil>=2.6.1->pandas) (1.12.0)
Installing collected packages: pandas
  Attempting uninstall: pandas
    Found existing installation: pandas 1.0.1
    Uninstalling pandas-1.0.1:
      Successfully uninstalled pandas-1.0.1
  Successfully installed pandas-1.0.3
WARNING: You are using pip version 20.1; however, version 20.1.1 is available.
You should consider upgrading via the 'c:\programdata\anaconda3\python.exe -m pip install --upgrade pip' command.
```

■ Can use Command Line in Jupyter/Colab

- Start with “!” = Command Line

Jupyter

```
!pip install numpy
```

```
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.16.5)
```

```
WARNING: You are using pip version 20.1; however, version 20.1.1 is available.
You should consider upgrading via the 'c:\programdata\anaconda3\python.exe -m pip install --upgrade pip' command.
```



Outlines

■ Basic Python

■ Python Data Types

- Basic/Primitive Data Types
- Collection Data Types

■ Condition

■ Loop

- For
- While

■ Function

■ Lambda Expression

■ Class

■ Python Packages

- Built-in Packages
- Import Other Packages
- Install Python Packages

■ NULL in Python

■ Reserved Words



NULL in Python

- **None** : NULL Object
- **NaN** (Not a Number) : NULL in number
- **Inf** : Infinity
- *** **None != NaN**

12 NULL

```
x = None  
print(x)  
print(type(x))
```

```
None  
<class 'NoneType'>
```

```
import numpy as np  
  
y = np.nan  
print(y)  
print(type(y))
```

```
nan  
<class 'float'>
```

Reserved Words

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

- สมชาย ประสิทธิ์จิตรระกุล, 2110101 Computer Programming, ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย
<https://www.cp.eng.chula.ac.th/~somchai/>
- Python for Data Science and Machine Learning Bootcamp <https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/>
- <http://cs231n.github.io/python-numpy-tutorial>



Course Review: Python for Data Science and Machine Learning Bootcamp



รศ. ดร. สมชาย ประสิทธิ์จิตรระกุล



Any Questions?