

Python Programming

COMP 8347

Slides prepared by Dr. Arunita Jaekel

arunita@uwindsor.ca



Python Basics

- Topics
 - If statements
 - Loops
 - Exception handling
 - Functions
 - File Input/Output
 - Modules



Control Flow

- Conditional branching
 - IF statement
- Looping
 - While
 - For ... in
- Exception handling
- Function or method call



IF Statement

- *suite*: a block of code, i.e. a sequence of one or more statements
- Syntax:
 - if bool_expression1:*
 - suite1*
 - elif bool_expression2:*
 - suite2*
 - ...
 - elif bool_expressionN:*
 - suiteN*
 - else:*
 - else_suite*
- No parenthesis or braces
 - used whenever a suite is to follow
 - Use indentation for block structure

```
if a < 10:
    print("few")
elif a < 25:
    print("some")
else:
    print("many")
```



While Statement

- Used to execute a suite 0 or more times
 - number of times depends on while loop's Boolean expression.
 - Syntax:

```
while bool_expression:  
    suite
```

- Example:

```
x, sum = 0,0  
while x < 10:  
    sum += x  
    x += 2  
print(sum, x)  #What is final value of sum and x
```

Answer: sum=**20**, x=**10**



For ... in Statement

- Syntax:

```
for variable in iterable:  
    suite
```

- Example: fruits = ['apple', 'pear', 'plum', 'peach']

```
for item in fruits:
```

```
    print(item)
```

- Alternatively

```
for i in range(len(fruits)):
```

```
    print(fruits[i])
```



Break and Continue

```
ex2.py - C:\Users\Arunita\OneDrive - University of Windsor\8347\slidesF20\ex2.py (3.8.3)
File Edit Format Run Options Window Help
1 # Example using break and continue
2 # Continue: control returns to top of current loop
3 # Break: exit from current loop
4
5 for num in [11, 8, 3, 25, 9, 16]:
6     if num > 20:
7         print('exiting loop')
8         break # exit the loop completely
9     elif num%2 == 0:
10        continue # immediately start next iteration
11    print(num)
12
```

Ln: 7 Col: 28

Iter#	num	num>20?	Num%2==0?	print(num)
0	11	n	n	11
1	8	n	y	
2	3	n	n	3
3	25	y		



Exception Handling

– Functions or methods indicate errors or other important events by ***raising exceptions***.

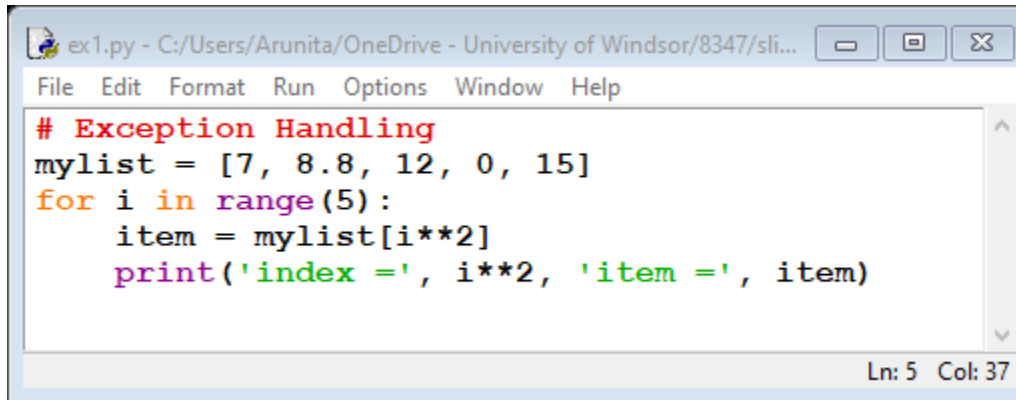
– Syntax (simplified):

```
try:  
    try_suite  
except exception1 as variable1:  
    exception_suite1  
  
...  
except exceptionN as variableN  
finally:  
    # cleanup
```

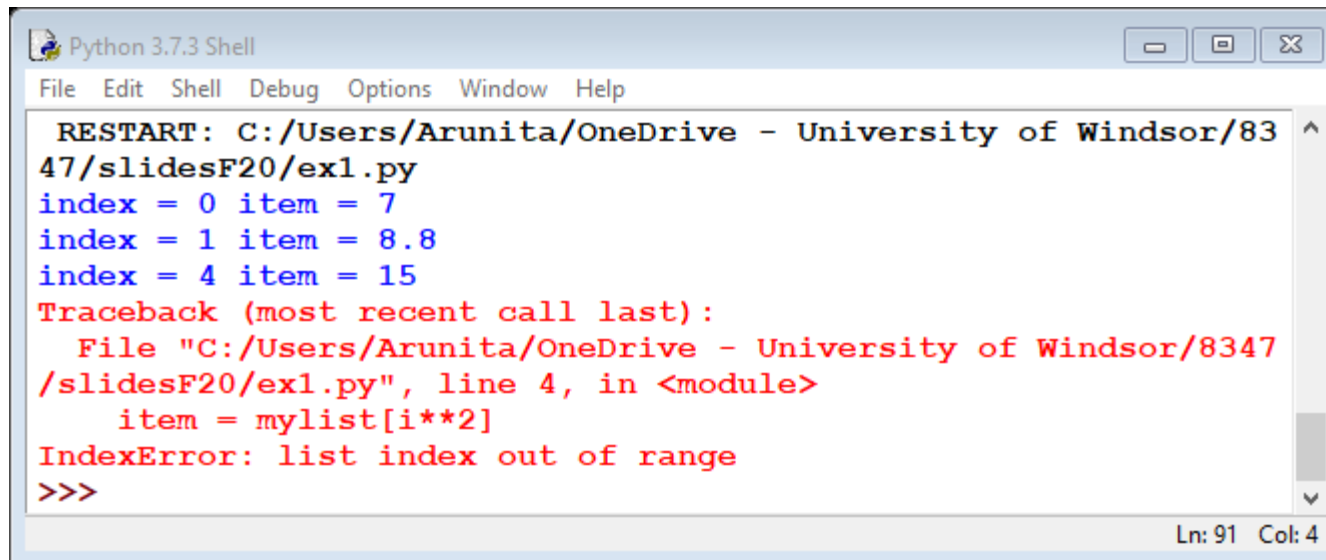
–variable part is optional



Exception Handling – Example 1

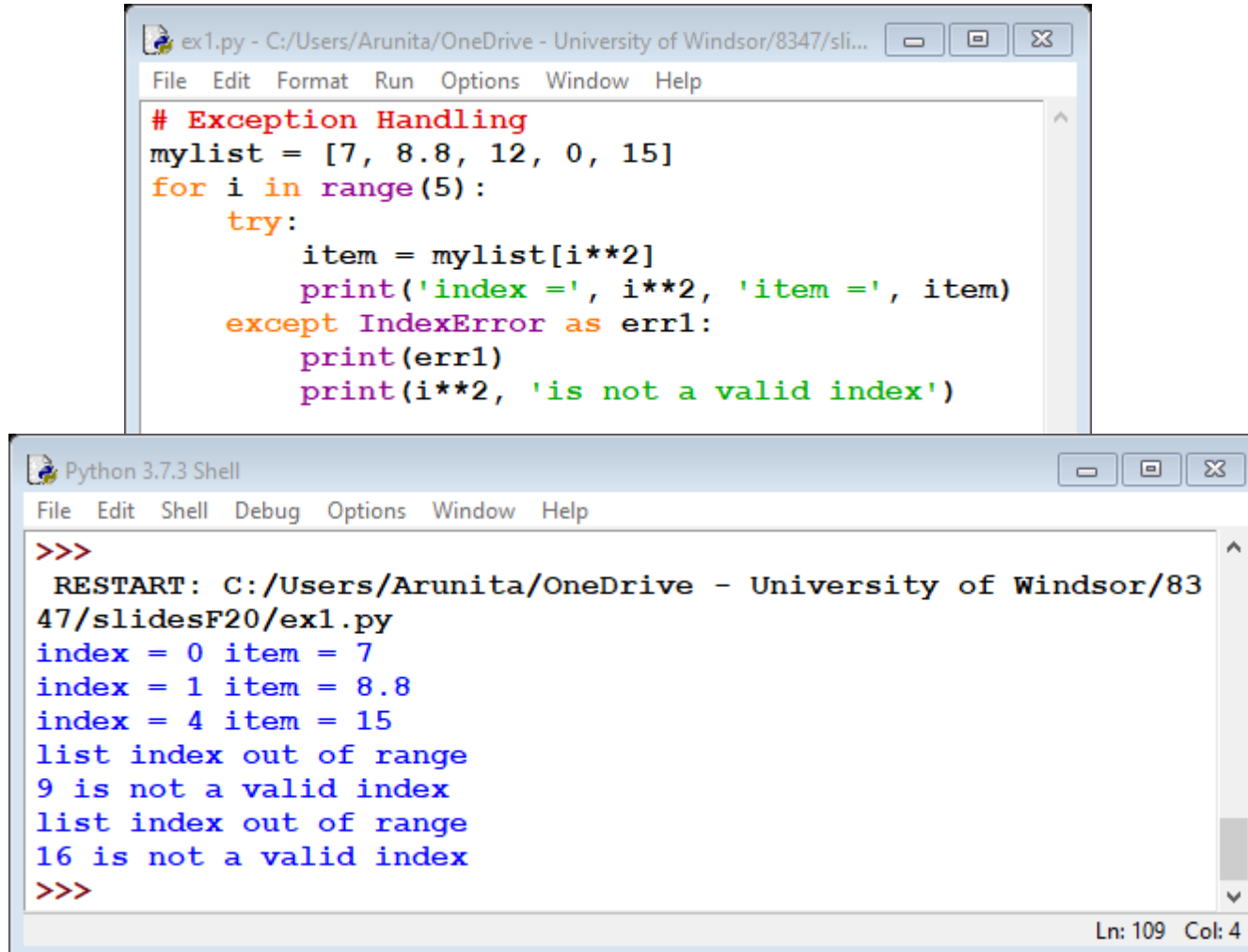


```
ex1.py - C:/Users/Arunita/OneDrive - University of Windsor/8347/sli...
File Edit Format Run Options Window Help
# Exception Handling
mylist = [7, 8.8, 12, 0, 15]
for i in range(5):
    item = mylist[i**2]
    print('index =', i**2, 'item =', item)
Ln: 5 Col: 37
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
RESTART: C:/Users/Arunita/OneDrive - University of Windsor/8347/slidesF20/ex1.py
index = 0 item = 7
index = 1 item = 8.8
index = 4 item = 15
Traceback (most recent call last):
  File "C:/Users/Arunita/OneDrive - University of Windsor/8347/slidesF20/ex1.py", line 4, in <module>
    item = mylist[i**2]
IndexError: list index out of range
>>>
Ln: 91 Col: 4
```

Exception Handling – Example 1



The image shows two windows from a Python IDE. The top window, titled 'ex1.py - C:/Users/Arunita/OneDrive - University of Windsor/8347/sli...', contains the following Python code:

```
# Exception Handling
mylist = [7, 8.8, 12, 0, 15]
for i in range(5):
    try:
        item = mylist[i*2]
        print('index =', i*2, 'item =', item)
    except IndexError as err1:
        print(err1)
        print(i*2, 'is not a valid index')
```

The bottom window, titled 'Python 3.7.3 Shell', shows the output of running the script. It starts with a restart message and then displays the results of the loop, including the error messages for the out-of-range indices.

```
>>>
RESTART: C:/Users/Arunita/OneDrive - University of Windsor/83
47/slidesF20/ex1.py
index = 0 item = 7
index = 1 item = 8.8
index = 4 item = 15
list index out of range
9 is not a valid index
list index out of range
16 is not a valid index
>>>
```

The status bar at the bottom right of the shell window indicates 'Ln: 109 Col: 4'.



Exception Handling – Example 2

Example:

```
s = input('Enter number: ')
try:
    n = float(s)
    print(n, ' is valid. ')
except ValueError as err:
    print(err)
```

- If user enters '8.6' output is: 8.6 is valid
- If user enters 'abc', output is:
ValueError: could not convert string to float: 'abc'



Functions

- Syntax:
def functionName(arguments):
suite
- Parameters are optional. Written as:
 - *positional arguments*: a sequence of comma separated identifiers
 - *keyword arguments*: a sequence of identifier=value pairs
- Every function returns a value
 - either a single value or tuple of values
 - return values can be ignored
 - can leave function at any point using *return* statement
 - no *return* statement or no argument for *return* → returns **None**



Functions

- Function definition:

```
def trianglePerimeter(s1, s2, s3):  
    perimeter = s1 + s2 + s3  
    return perimeter
```

- **Sample function call: sides = [5, 8, 2]**

- `result = trianglePerimeter(sides[0], sides[1], sides[2])`
 - `result = trianglePerimeter(*sides)` # use sequence unpacking operator (*)

- **General function for any n-sided polygon:**

```
def perimeter(* sides):  
    result = 0  
    for s in sides:  
        result += s  
    return result
```



File Input/Output

- `f = open(filename, mode)`
 - mode is optional; possible values:
 - `'w'` = write
 - `'r'` = read (default)
 - `'a'` = append
 - `'rb'` (`'wb'`) = read (write) in binary
 - Example:
 - `f1 = open("infile.txt")`
 - `f2 = open("outfile.txt", "w")`
 - `f` is iterable:

```
for x in f:  
    print(x)
```



File Methods

- *f.close()*: closes file object *f*
- *f.peek(n)*: returns *n* bytes without moving file pointer position
- *f.read(n)*: read at most *n* bytes from *f*
- *f.read()*: read every byte starting from current position
- *f.readline()*: read next line
- *f.readlines()*: read all the lines to the end of file and return them as a list
- *f.write(s)*: write byte/bytearray object *s* to *f* in binary mode str object *s* in text mode.
- *f.writelines(seq)*: write the sequence of objects (strings or byte strings, depending on mode) to *f*.



Modules

- *Modules*: Contains additional functions and custom data types.
 - Usually a .py file containing Python code; can be written in other languages
 - designed to be imported and used by other programs
- *Packages*: Sets of modules that are grouped together.
- Usage examples:
 - `import os`
 - `import math`
 - `from os import path`



Summary

- Control flow statements
- File I/O
- Modules

