METU Computer Engineering

CEng 240 – Spring 2021 Week 12

Sinan Kalkan

Error Handling & Debugging

Disclaimer: Figures without reference are from either from "Introduction to programming concepts with case studies in Python" or "Programming with Python for Engineers", which are both co-authored by me.

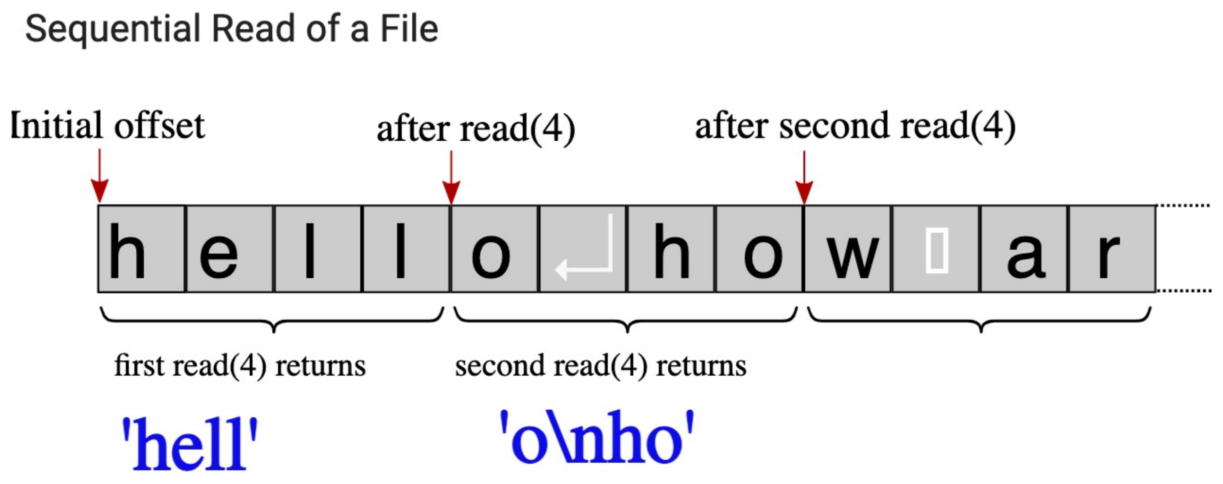
Previously on CENG240!

Opening/closing files

- Opening files:
  - open(filename, “r”) => open file for reading
  - open(filename, “w”) => open file for writing
  - open(filename, “a”) => open file for appending
- Closing file:
  - fileobject.close()

Previously on CENG240!

Files and Sequential Access



```
fp = open("firstexample.txt","r") # the example file we created above

for i in range(3): # repeat 3 times
        content = fp.read(4) # read 4 bytes in each step
        print("> ", content) # output 4 bytes preceded by >

fp.close()
```

Previously on CENG240!

Accessing Files Line by Line

```
pointlist = [(0,0), (10,0), (10,10), (0,10)]

fp = open("pointlist.txt", "w") # open file for writing
fp.write(str(len(pointlist))) # write list length
fp.write('\n’)

# Go over each point in the list
for (x,y) in pointlist: # for each x,y value in the list
        fp.write(str(x)) # write x
        fp.write(' ') # space as number separator
        fp.write(str(y)) # write y
        fp.write('\n') # \n as line separator

fp.close()
```

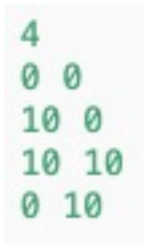
Produces file with content:

```
4
0 0
10 0
10 10
0 10
```

Previously on CENG240!

Accessing Files Line by Line

Read file with content:



```
4
0 0
10 0
10 10
0 10
```

```
fp = open("pointlist.txt") # open file for reading

nextline = fp.readline() # read the first line
while nextline != '': # while read is successful
        print(nextline) # output the line
        nextline = fp.readline() # read the nextline

fp.close() # when nextline == '' loop terminates
```

Previously on CENG240!

Termination of input

- There are two ways to stop reading input:
  1. By reading a definite number of items.
     - Call read() or readline() functions for a fixed number of times.
  2. By the end of the file.
     - Continue to read() or readline() until they return empty string ‘’.

```
fp = open("pointlist.txt") # open file for reading

nextline = fp.readline() # read the first line
while nextline != '': # until end of file
        ... # Do something  with the read line
        nextline = fp.readline() # read the next line
```

This Week

- Error handling and Debugging:
  - Kinds of errors
  - Exceptions
  - Debugging techniques

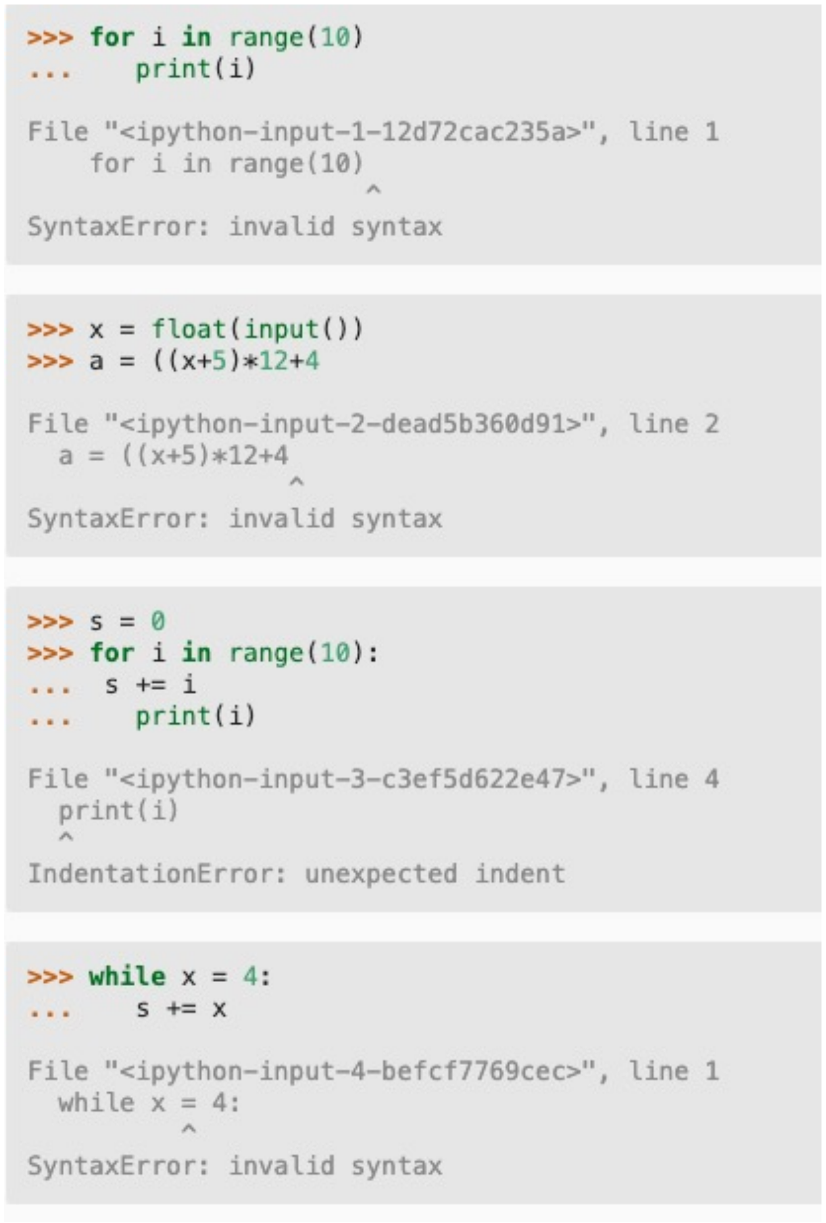
Administrative Notes

- Lab 8
- ~~Midterm: 1 June, Tuesday, 17:40~~

METU Computer Engineering

Error Types

- Syntax errors
- Type errors
- Run-time errors
- Logical errors



```
>>> for i in range(10)
...     print(i)

File "<ipython-input-1-12d72cac235a>", line 1
    for i in range(10)
                      ^
SyntaxError: invalid syntax
```

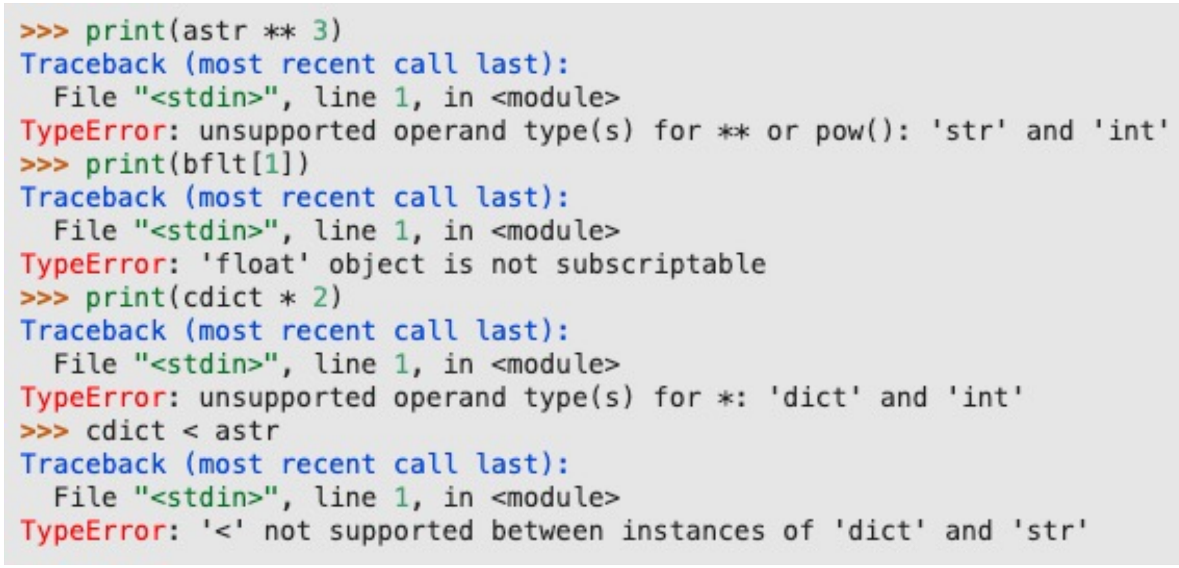
```
>>> x = float(input())
>>> a = ((x+5)*12+4

File "<ipython-input-2-dead5b360d91>", line 2
  a = ((x+5)*12+4
                 ^
SyntaxError: invalid syntax
```

```
>>> s = 0
>>> for i in range(10):
...  s += i
...     print(i)

File "<ipython-input-3-c3ef5d622e47>", line 4
  print(i)
  ^
IndentationError: unexpected indent
```

```
>>> while x = 4:
...     s += x

File "<ipython-input-4-befcf7769cec>", line 1
  while x = 4:
          ^
SyntaxError: invalid syntax
```

Error Types

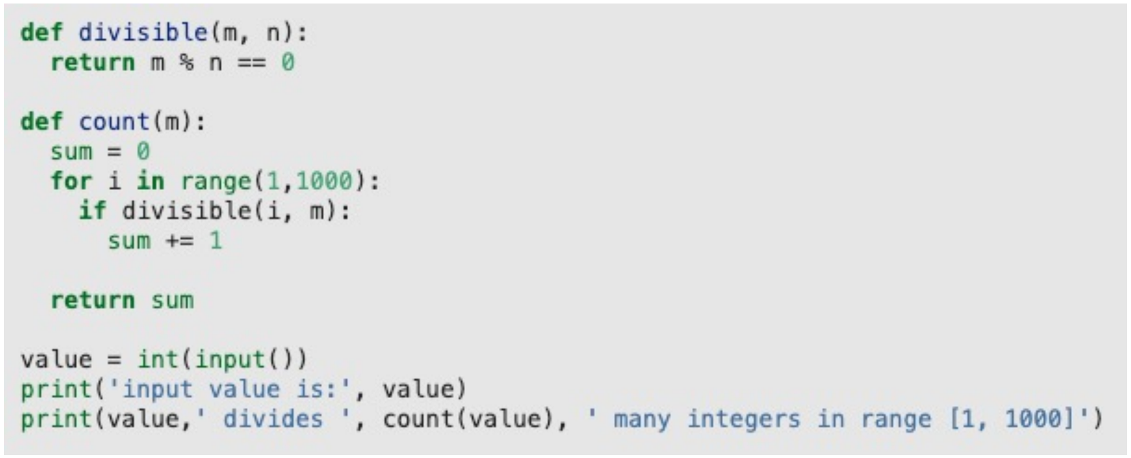
- Syntax errors
- Type errors
- Run-time errors
- Logical errors



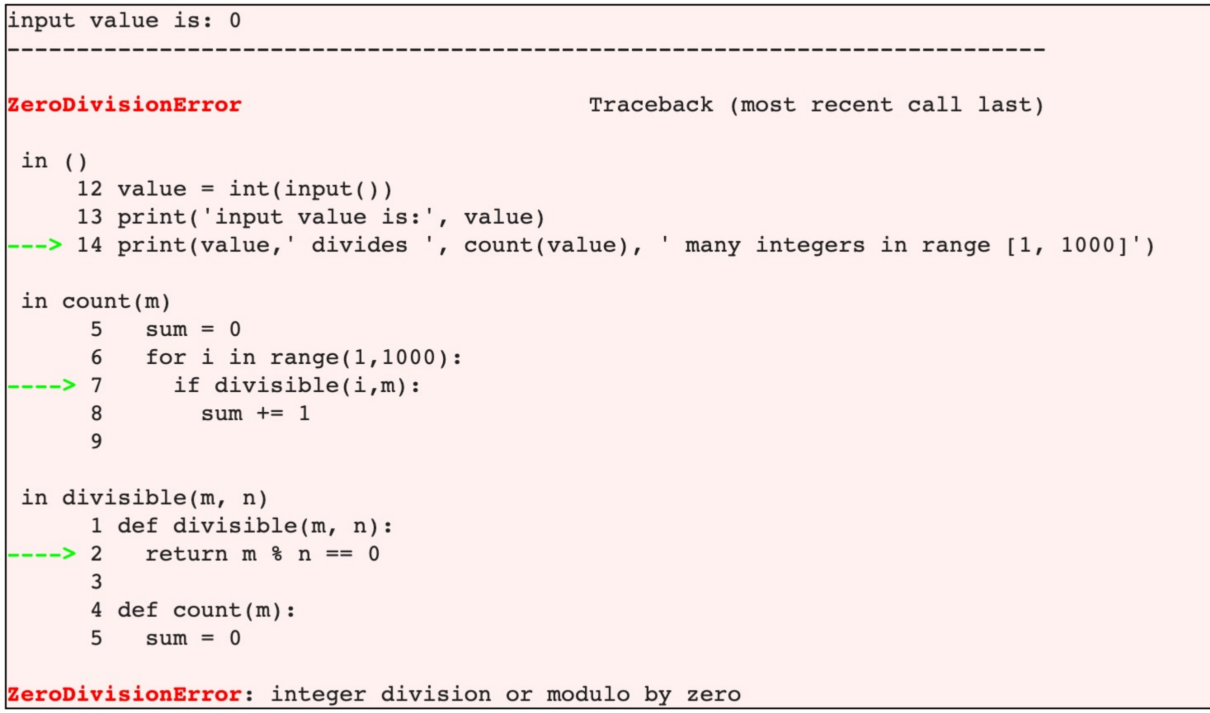
```
>>> print(astr ** 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
>>> print(bflt[1])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object is not subscriptable
>>> print(cdict * 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for *: 'dict' and 'int'
>>> cdict < astr
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'dict' and 'str'
```

Error Types

- Syntax errors
- Type errors
- Run-time errors
- Logical errors

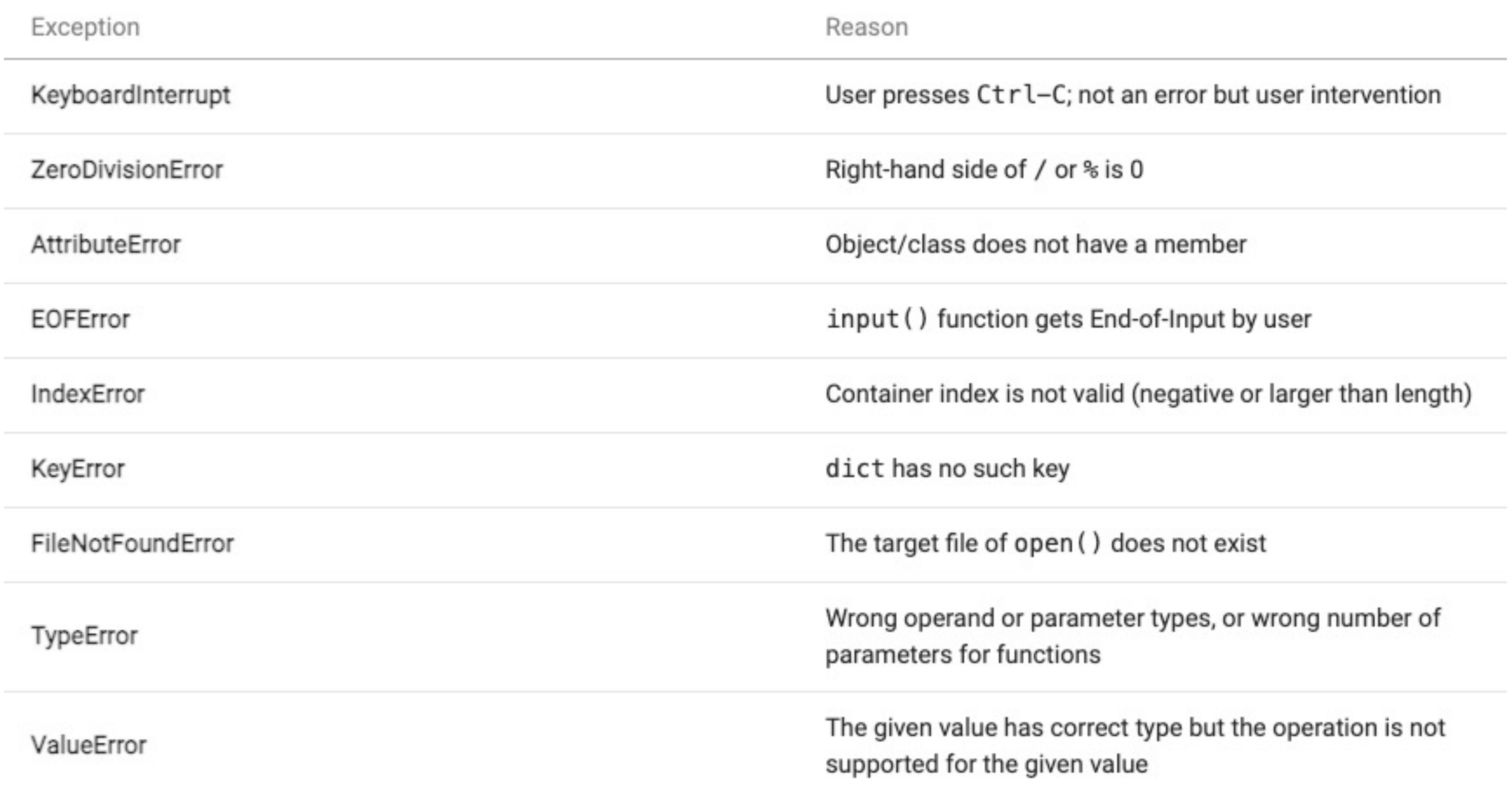


```python
def divisible(m, n):
  return m % n == 0

def count(m):
  sum = 0
  for i in range(1,1000):
    if divisible(i, m):
      sum += 1

  return sum

value = int(input())
print('input value is:', value)
print(value,' divides ', count(value), ' many integers in range [1, 1000]')
```



```
input value is: 0
---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call last)

 in ()
     12 value = int(input())
     13 print('input value is:', value)
---> 14 print(value,' divides ', count(value), ' many integers in range [1, 1000]')

 in count(m)
      5   sum = 0
      6   for i in range(1,1000):
----> 7     if divisible(i,m):
      8       sum += 1
      9

 in divisible(m, n)
      1 def divisible(m, n):
----> 2   return m % n == 0
      3
      4 def count(m):
      5   sum = 0

ZeroDivisionError: integer division or modulo by zero
```

Exceptions



Exception	Reason
KeyboardInterrupt	User presses `Ctrl-C`; not an error but user intervention
ZeroDivisionError	Right-hand side of / or % is 0
AttributeError	Object/class does not have a member
EOFError	`input()` function gets End-of-Input by user
IndexError	Container index is not valid (negative or larger than length)
KeyError	`dict` has no such key
FileNotFoundError	The target file of `open()` does not exist
TypeError	Wrong operand or parameter types, or wrong number of parameters for functions
ValueError	The given value has correct type but the operation is not supported for the given value

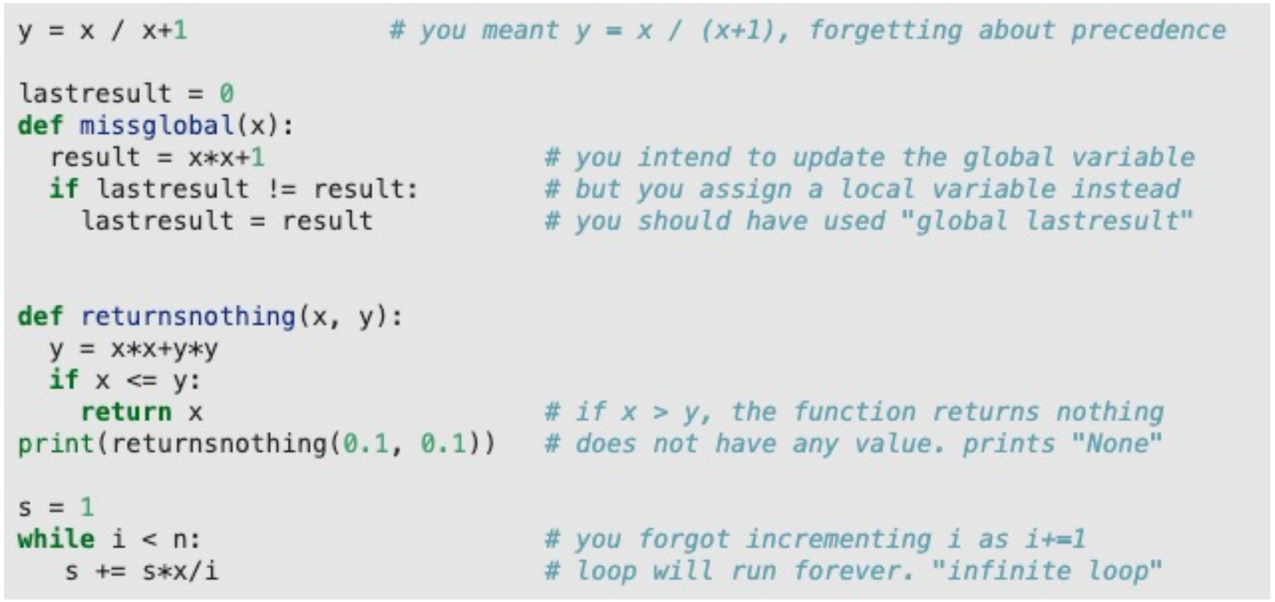
METU Computer Engineering

Exception Examples

```
b = 0
a = a / b                  # ZeroDivisionError

x = [1,2,3]
print(x.length)            # AttributeError: lists does not have a length attribute (use len(x))

print(x[4])                # IndexError: last valid index of list x is 2

person = { 'name' : 'Han', 'surname': 'Solo'}
print(person['Name'])      # KeyError: person does not have 'Name' key but 'name'

fp = open("example.txt") # FileNotFoundError: file "example.txt" does not exist

print([1,2,3] / 12)        # TypeError: Division is not defined for lists

def f(x, y):
  return x*x+y

print(f(5))                # TypeError: Only one element is supplied instead of 2.

print(int('thirtytwo'))  # ValueError: string value does not represent an integer

a,b,c = [1,2,3,4]          # ValueError: too many values on the right hand side
```

Error Types

- Syntax errors
- Type errors
- Run-time errors
- Logical errors



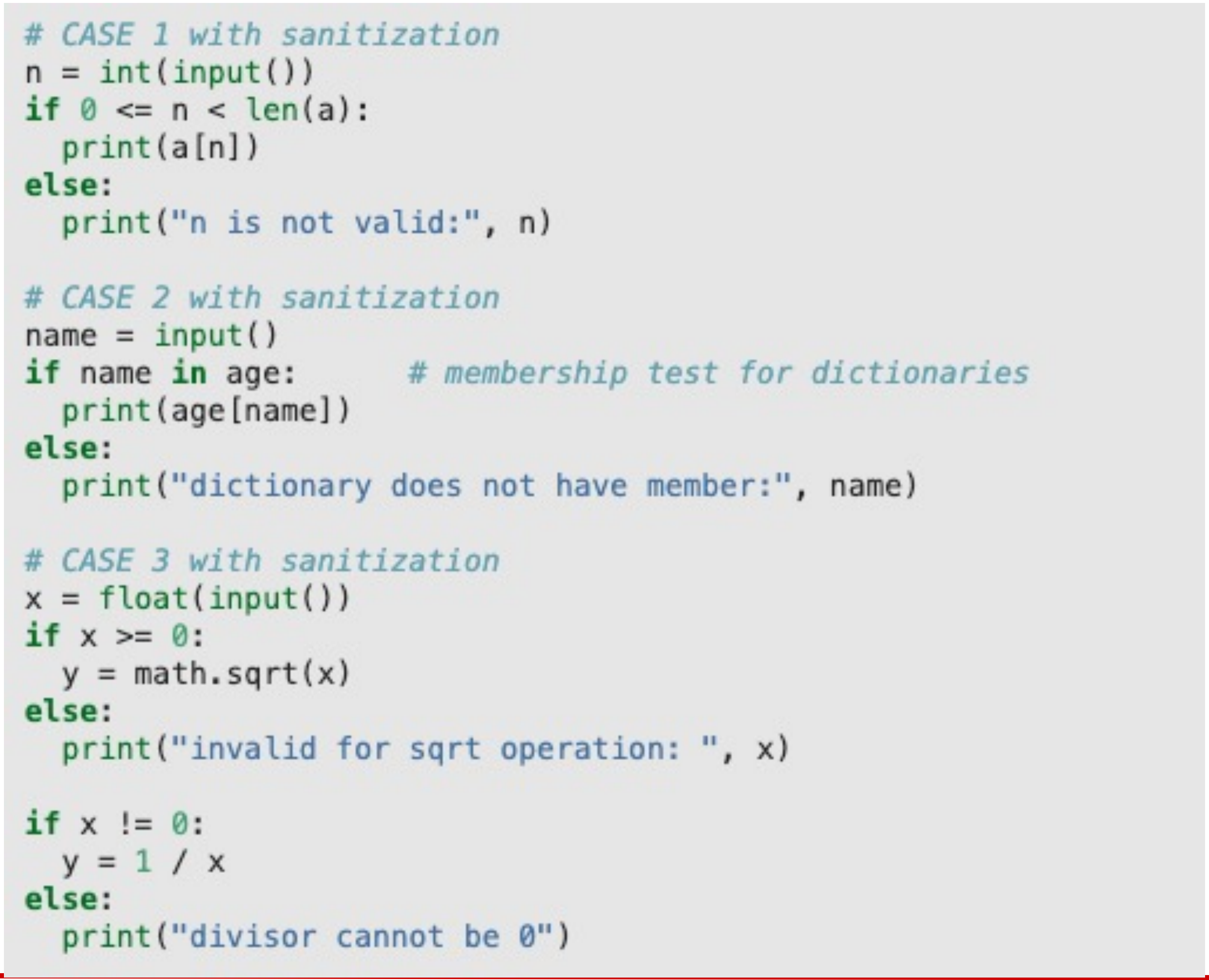
```
y = x / x+1               # you meant y = x / (x+1), forgetting about precedence

lastresult = 0
def missglobal(x):
  result = x*x+1                    # you intend to update the global variable
  if lastresult != result:          # but you assign a local variable instead
    lastresult = result             # you should have used "global lastresult"


def returnsnothing(x, y):
  y = x*x+y*y
  if x <= y:
    return x                        # if x > y, the function returns nothing
print(returnsnothing(0.1, 0.1))     # does not have any value. prints "None"

s = 1
while i < n:                        # you forgot incrementing i as i+=1
   s += s*x/i                       # loop will run forever. "infinite loop"
```

How to work with errors

1. Program with care
2. Place controls in your code
3. Handle exceptions
4. Write verification code & raise exceptions
5. Debug your code
6. Write test cases

How to work with errors:

(2) Place controls in your code



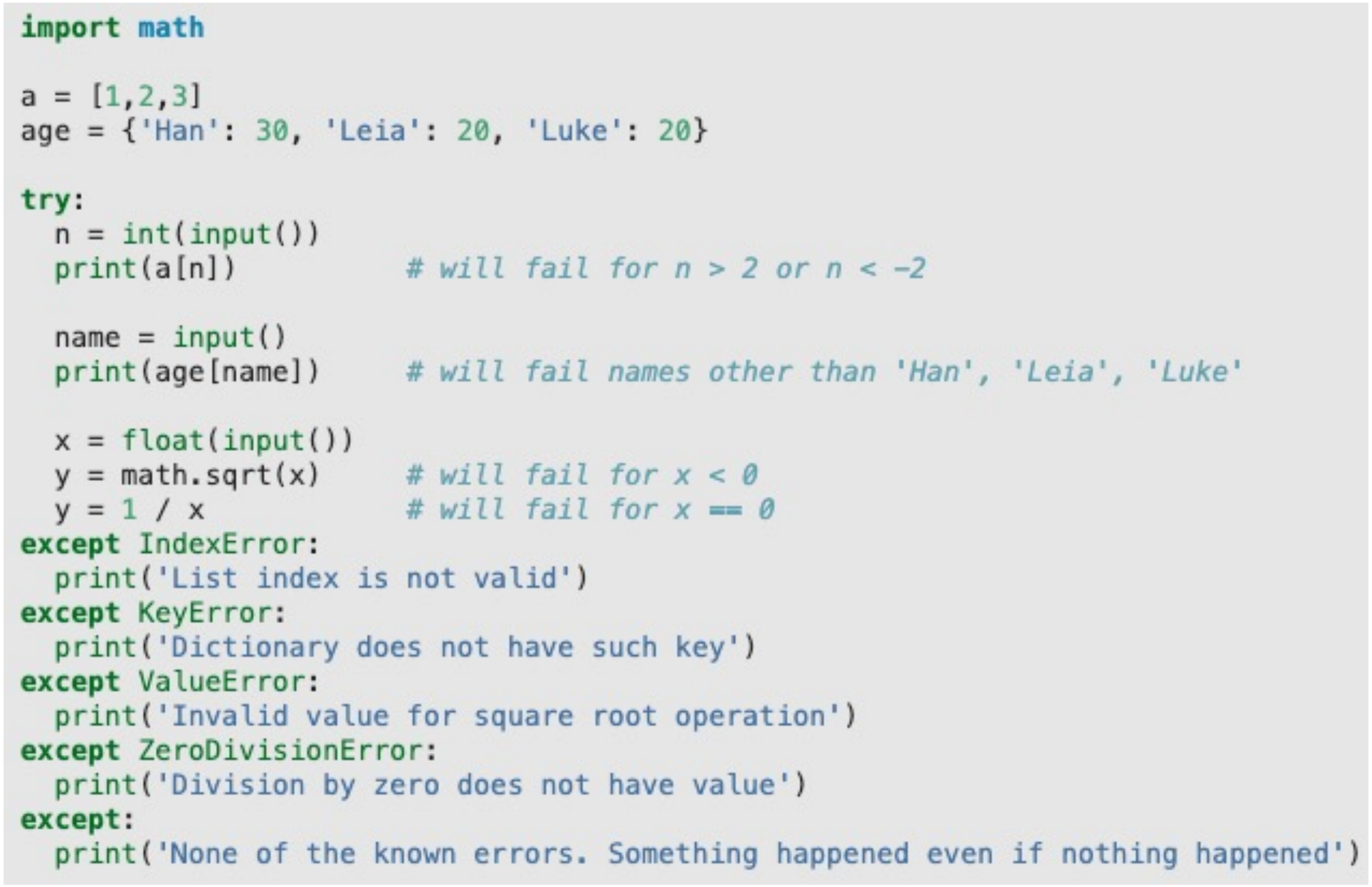
```python
# CASE 1 with sanitization
n = int(input())
if 0 <= n < len(a):
  print(a[n])
else:
  print("n is not valid:", n)

# CASE 2 with sanitization
name = input()
if name in age:        # membership test for dictionaries
  print(age[name])
else:
  print("dictionary does not have member:", name)

# CASE 3 with sanitization
x = float(input())
if x >= 0:
  y = math.sqrt(x)
else:
  print("invalid for sqrt operation: ", x)

if x != 0:
  y = 1 / x
else:
  print("divisor cannot be 0")
```

How to work with errors:
(3) Handle Exceptions

```
try:
    ......      # a block with possible errors
    ......      # if there are function calls here
    ......      # and error occurs in the function, we can handle error here
except exceptionname:       # exceptionname is optional
    .....       # this is error handling block.
    .....       # when there is an error, execution jumps here
```

How to work with errors:

(3) Handle Exceptions



```
import math

a = [1,2,3]
age = {'Han': 30, 'Leia': 20, 'Luke': 20}

try:
  n = int(input())
  print(a[n])           # will fail for n > 2 or n < -2

  name = input()
  print(age[name])      # will fail names other than 'Han', 'Leia', 'Luke'

  x = float(input())
  y = math.sqrt(x)      # will fail for x < 0
  y = 1 / x             # will fail for x == 0
except IndexError:
  print('List index is not valid')
except KeyError:
  print('Dictionary does not have such key')
except ValueError:
  print('Invalid value for square root operation')
except ZeroDivisionError:
  print('Division by zero does not have value')
except:
  print('None of the known errors. Something happened even if nothing happened')
```

How to work with errors:

(4) Write verification code and raise exception

- You can raise exceptions
- “raise Exception” => raise a generic exception

```
try:
  if !cond1:
    raise Error

  ..1..

  if !cond2:
    raise Error

  ..2..

  if !cond3:
    raise Error

  ..3..
  ..4..   # success
except :
  ... Error handling
```

```
def solvesecond(a,b,c):
  det = b*b - 4*a*c
  # the following is the verification code
  if det < 0:
    print("Equation has no real roots for", a, b, c)
    raise ValueError
  ....
  ...
```

How to work with errors:

(6) Write test cases

```
(x1, x2) = findrootsecond(a,b,c)

if a*x1*x1 + b*x1 + c != 0 or a*x2*X2 + b*x2 +c != 0:
   print('test failed for', a, b, c, 'roots', x1, x2)
```

Debugging

- Using debugging outputs
- Handling exception and getting more info
- Using debugger

Debugging: Using debugging outputs

- The following code has a bug, how can we find it?

```
def sum_and_delete(L):
        sum = 0
        for i in range(len(L)):
                sum += L[i]
                del L[i]

        return sum

sum_and_delete([1, 2, 3, 4, 5])
```

Debugging:
Handling Exception to Get More Info

```
def sum_and_delete(L):
        sum = 0
        try:
                for i in range(len(L)):
                        sum += L[i]
                        del L[i]
        except:
                print(f"i: {i} len(L): {len(L)}")

        return sum

sum_and_delete([1, 2, 3, 4, 5])
```

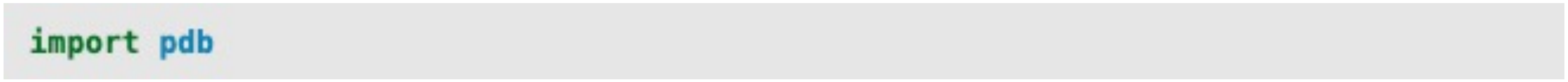
Debugging:
Using debugger

```
import pdb
```

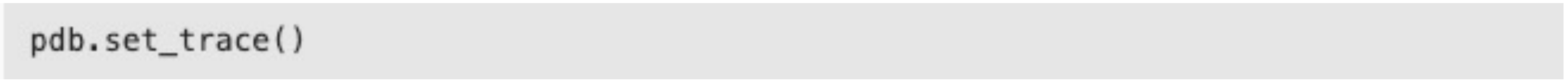
```
pdb.set_trace()
```

```
> <ipython-input-14-110393975fb5>(7)startswith()
-> for i in range(len(srcstr)): # check all characters of srcstr
(Pdb) h

Documented commands (type help <topic>):
========================================
EOF    c          d        h         list      q        rv       undisplay
a      cl         debug    help      ll        quit     s        unt
alias  clear      disable  ignore    longlist  r        source   until
args   commands   display  interact  n         restart  step     up
b      condition  down     j         next      return   tbreak   w
break  cont       enable   jump      p         retval   u        whatis
bt     continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:
==========================
exec  pdb
```

Debugging:
Using debugger



```
import pdb
```



```
pdb.set_trace()
```

```
import pdb

def sum_and_delete(L):
        sum = 0
        pdb.set_trace()
        for i in range(len(L)):
                sum += L[i]
                del L[i]

        return sum

sum_and_delete([1, 2, 3, 4, 5])
```

Example

- Calculate Discount Averages

https://pp4e-workbook.github.io/chapters/error_handling_and_debugging/calculate_discount_averages.html

- Memento

https://pp4e-workbook.github.io/chapters/error_handling_and_debugging/memento.html

METU Computer Engineering

Final Words: Important Concepts

- Different types of errors: Syntax, type, run-time and logical errors.
- How to deal with errors.
- Exceptions and exception handling.
- Debugging by “printing” values, exception handling and a debugger.

THAT’S ALL FOLKS!
STAY HEALTHY