

# The Python Language (Part 2)

Copyright © 2024 by  
Robert M. Dondero, Ph.D.  
Princeton University

# Objectives

- We will cover:
  - A subset of Python...
  - That is appropriate for COS 333...
  - Through example programs

# Agenda

- **Data types and operators**
- **Terminal I/O**
- Catching exceptions
- Statements
- Throwing exceptions

# Data Types and Operators

- See **circle1.py**

```
$ python circle1.py
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415927.
$ python circle1.py
Enter the circle's radius:
1
A circle with radius 1 has diameter 2
and circumference 6.283185.
$
```

`__name__` is pronounced “dunder name”  
`__main__` is pronounced “dunder main”  
`__o__` is pronounced ???

# Data Types and Operators

Data Type	Example Literals
int	1, 23, 3493, 01, 027, 06645, 0x1, 0x17, 0xDA5 <b>(no theoretical size limit)</b>
float	0., 0.0, .0, 1.0, 1e0, 1.0e0 <b>(corresponds to C/Java double)</b>
bool	False, True
str	'hi', "hi"

# Data Types and Operators

<b>Conversion Function</b>	<b>Usage</b>
<code>int(object)</code>	frequent
<code>float(object)</code>	frequent
<code>bool(object)</code>	infrequent
<code>str(object)</code>	frequent

# Data Types and Operators

Operator	(Priority) Meaning
<code>{key: expr, ...} , {expr, ...}</code>	(1) <b>Dictionary</b> creation; <b>set</b> creation
<code>[expr, ...]</code>	(2) <b>List</b> creation
<code>(expr, ...)</code>	(3) <b>Tuple</b> creation, or just parentheses
<code>f(expr, ...)</code>	(4) Function call
<code>x[startindex:stopindex]</code>	(5) Slicing (for sequences)
<code>x[index]</code>	(6) Indexing (for containers)
<code>x.attr</code>	(7) Attribute reference
<code>x**y</code>	(8) Exponentiation
<code>~x</code>	(9) Bitwise NOT
<code>+x, -x</code>	(10) Unary plus, unary minus
<code>x*y, x/y, x//y, x%y</code>	(11) Mult, div, truncating div, remainder (or string formatting)
<code>x+y, x-y</code>	(12) Addition, subtraction

# Data Types and Operators

Beware of division operators:

Expression	Value
<b>5 // 2</b>	<b>2</b>
5 / 2	2.5
<b>float(5) / float(2)</b>	2.5
4 / 2	2.0
<b>float(4) / float(2)</b>	2.0

Suggestion:  
use only  
boldfaced  
forms



# Data Types and Operators

Operator	(Priority) Meaning
<code>x&lt;&lt;i, x&gt;&gt;i</code>	(13) Left-shift, right-shift
<code>x&amp;y</code>	(14) Bitwise AND
<code>x^y</code>	(15) Bitwise XOR
<code>x y</code>	(16) Bitwise OR
<code>x&lt;y, x&lt;=y, x&gt;y, x&gt;=y</code>	(17) Relational
<code>x==y, x!=y</code>	(17) Relational
<code>x is y, x is not y</code>	(18) <b>Identity</b> tests
<code>x in y, x not in y</code>	(19) <b>Membership</b> tests
<code>not x</code>	(20) Logical NOT
<code>x and y</code>	(21) Logical AND
<code>x or y</code>	(22) Logical OR

# Terminal I/O

## Reading from stdin:

```
str = input()
str = input(prompt_str)

import sys
...
str = sys.stdin.readline()
```

# Terminal I/O

## Writing to stdout:

```
print(str)
print(str, end='')
print(str1, str2, str3)
print(str1, str2, str3, end='')
```

## Writing to stderr:

```
import sys
...
print(str, file=sys.stderr)
print(str, end='', file=sys.stderr)
print(str1, str2, str3, end='', file=sys.stderr)
```

# Agenda

- Data types and operators
- Terminal I/O
- **Catching exceptions**
- Statements
- Throwing exceptions

# Catching Exceptions

- Recall **circle1.py**

```
$ python circle1.py
Enter the circle's radius:
xyz
Traceback (most recent call last):
  File "circle1.py", line 26, in <module>
    main()
  File "circle1.py", line 15, in main
    radius = int(line)
ValueError: invalid literal for int() with base 10: 'xyz'
$ python circle1.py
Enter the circle's radius:
Traceback (most recent call last):
  File "circle1.py", line 26, in <module>
    main()
  File "circle1.py", line 14, in main
    line = input("Enter the circle's radius:\n")
EOFError
$
```

# Catching Exceptions

- See [circle2.py](#)

```
$ python circle2.py
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415927.
$ python circle2.py
Enter the circle's radius:
xyz
invalid literal for int() with base 10: 'xyz'
$ python circle2.py
Enter the circle's radius:

$
```

# Catching Exceptions

- See **circle3.py**

```
$ python circle3.py
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415927.
$ python circle3.py
Enter the circle's radius:
xyz
Error: Not an integer
$ python circle3.py
Enter the circle's radius:
Error: Missing integer
$
```

# Catching Exceptions

- See [circle4.py](#)

```
$ python circle4.py
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415927.
$ python circle4.py
Enter the circle's radius:
xyz
Error: Not an integer
$ python circle4.py
Enter the circle's radius:
Error: Missing integer
$
```



# Catching Exceptions

```
try:
    stmt1
    stmt2
    stmt3
except ExceptionClass1:
    stmt4
    stmt5
    stmt6
except ExceptionClass2:
    stmt7
    stmt8
    stmt9

stmt10
stmt11
stmt12
```

## Case 1:

Python executes *stmt1*,  
*stmt2*, *stmt3* successfully

# Catching Exceptions

```
try:
    stmt1
    stmt2
    stmt3
except ExceptionClass1:
    stmt4
    stmt5
    stmt6
except ExceptionClass2:
    stmt7
    stmt8
    stmt9
stmt10
stmt11
stmt12
```

## Case 2:

stmt2 throws an object of some class that matches *ExceptionClass1*

The thrown object **matches** *ExceptionClass1* if the class of the thrown object is *ExceptionClass1* or any subclass of *ExceptionClass1*

# Catching Exceptions

```
try:
    stmt1
    stmt2
    stmt3
except ExceptionClass1:
    stmt4
    stmt5
    stmt6
except ExceptionClass2:
    stmt7
    stmt8
    stmt9
stmt10
stmt11
stmt12
```

## **Case 3:**

stmt2 throws an object of some class that does not match *ExceptionClass1*, but does match *ExceptionClass2*

# Catching Exceptions

```
try:
    stmt1
    stmt2
    stmt3
except ExceptionClass1:
    stmt4
    stmt5
    stmt6
except ExceptionClass2:
    stmt7
    stmt8
    stmt9
stmt10
stmt11
stmt12
```

## **Case 4:**

stmt2 throws an object of some class that matches neither *ExceptionClass1* nor *ExceptionClass2*

Python propagates the exception outward and upward, and repeats the algorithm at each level

# Aside: Exit Status

- See circle5.py

```
$ python circle5.py
Enter the circle's radius:
5
A circle with radius 5 has diameter 10
and circumference 31.415927.
$ echo $?
0
$ python circle5.py
Enter the circle's radius:
xyz
Error: Not an integer
$ echo $?
1
$ python circle5.py
Enter the circle's radius:
Error: Missing integer
$ echo $?
1
$
```

On MS Windows use:  
**echo %errorlevel%**

# Agenda

- Data types and operators
- Terminal I/O
- Catching exceptions
- **Statements**
- Throwing exceptions

# Statements

- See **euclidclient1.py**

```
$ python euclidclient1.py
Enter the first integer: 8
Enter the second integer: 12
gcd: 4
lcm: 24
$
```

abs is defined in the `builtins` module

All names defined in the `builtins` module are imported automatically, as if via `from builtins import *`

# Statements

- See **euclidclient1.py** (cont.)
  - *Unpacking assignment statement*

```
x, y = 1, 2
```

## Traditional

```
temp = i%j  
i = j  
j = temp
```

## Verbose

```
temp1 = j  
temp2 = i%j  
i = temp1  
j = temp2
```

## Python

```
i, j = j, i%j
```



# Statements

## Assignment statements

*var = expr*

*var += expr*

*var -= expr*

*var \*= expr*

*var /= expr*

*var // = expr*

*var %= expr*

*var \*\* = expr*

*var & = expr*

*var |= expr*

*var ^= expr*

*var >> = expr*

*var << = expr*

# Statements

Unpacking assignment statement

```
var1, var2, ... = iterable
```

No-op statement

```
pass
```

assert **statement**

```
assert expr, message
```

# Statements

Function call statement

```
f(expr, name=expr, ...)
```

return **statement**

```
return
```

```
return expr
```

# Statements

```
if statement
    if expr:
        statement (s)
    elif expr:
        statement (s)
    ...
    else:
        statement (s)
```

False, 0, None, '', "", [], (), and {}  
indicate logical FALSE  
Any other value indicates logical TRUE

# Statements

```
while statement  
    while expr:  
        statement (s)
```

False, 0, None, ' ', "", [], (), and {}  
indicate logical FALSE  
Any other value indicates logical TRUE

# Statements

```
for statements  
    for var in iterable:  
        statement(s)
```

```
break statement  
    break
```

```
continue statement  
    continue
```

```
for i in range(0, 5):  
    ... i ...  
for i in range(5):  
    ... i ...
```

# Statements

```
try statement
    try:
        statement(s)
    except [ExceptionType [as var]]:
        statement(s)

raise statement
    raise ExceptionType(str)
```

# Agenda

- Data types and operators
- Terminal I/O
- Catching exceptions
- Statements
- **Throwing exceptions**



# Throwing Exceptions

- Recall **euclidclient1.py**

```
$ python euclidclient1.py
Enter the first integer: 0
Enter the second integer: 12
gcd: 12
lcm: 0
$ python euclidclient1.py
Enter the first integer: 0
Enter the second integer: 0
gcd: 0
Traceback (most recent call last):
  File "euclidclient1.py", line 53, in <module>
    main()
  File "euclidclient1.py", line 42, in main
    my_lcm = lcm(i, j)
  File "euclidclient1.py", line 26, in lcm
    return (i // gcd(i, j)) * j
ZeroDivisionError: integer division or modulo by zero
$
```

# Throwing Exceptions

- See **euclidclient2.py**

```
$ python euclidclient2.py
Enter the first integer: 8
Enter the second integer: 12
gcd: 4
lcm: 24
$ python euclidclient2.py
Enter the first integer: 0
Enter the second integer: 12
gcd: 12
lcm(i,j) is undefined if i or j is 0
$ python euclidclient2.py
Enter the first integer: 0
Enter the second integer: 0
gcd(i,j) is undefined if i and j are 0
$
```

# Throwing Exceptions

```
BaseException
  Exception
    ArithmeticError
      FloatingPointError
      OverflowError (legacy)
      ZeroDivisionError
    AssertionError
    AttributeError
    BufferError
    EOFError
    ImportError
      ModuleNotFoundError
    LookupError
      IndexError
      KeyError
    MemoryError
    NameError
      UnboundLocalError
```

Python  
standard  
exception  
classes

# Throwing Exceptions

```
BaseException (cont.)
  Exception (cont.)
    OSError
      BlockingIOError
      ChildProcessError
      ConnectionError
        BrokenPipeError
        ConnectionAbortedError
        ConnectionRefusedError
        ConnectionResetError
      FileExistsError
      FileNotFoundError
      InterruptedError
      IsADirectoryError
      NotADirectoryError
      PermissionError
      ProcessLookupError
      TimeoutError
```

Python  
standard  
exceptions  
(cont.)

# Throwing Exceptions

```
BaseException (cont.)
  Exception (cont.)
    ReferenceError
    RuntimeError
      NotImplementedError
      RecursionError
    StopIteration
    StopAsyncIteration
    SyntaxError
      IndentationError
      TabError
    SystemError
    TypeError
    ValueError
      UnicodeError
        UnicodeDecodeError
        UnicodeEncodeError
        UnicodeTranslateError
```

Python  
standard  
exceptions  
(cont.)

# Throwing Exceptions

```
BaseException (cont.)
  Exception (cont.)
    Warning
      BytesWarning
      DeprecationWarning
      FutureWarning
      ImportError
      PendingDeprecationWarning
      ResourceWarning
      RuntimeWarning
      SyntaxWarning
      UnicodeWarning
      UserWarning
    GeneratorExit
    KeyboardInterrupt
    SystemExit
```

Python  
standard  
exceptions  
(cont.)

# Summary

- We have covered these aspects of Python:
  - Data types and operators
  - Terminal I/O
  - Catching exceptions
  - Statements
  - Throwing exceptions