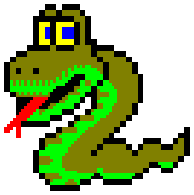
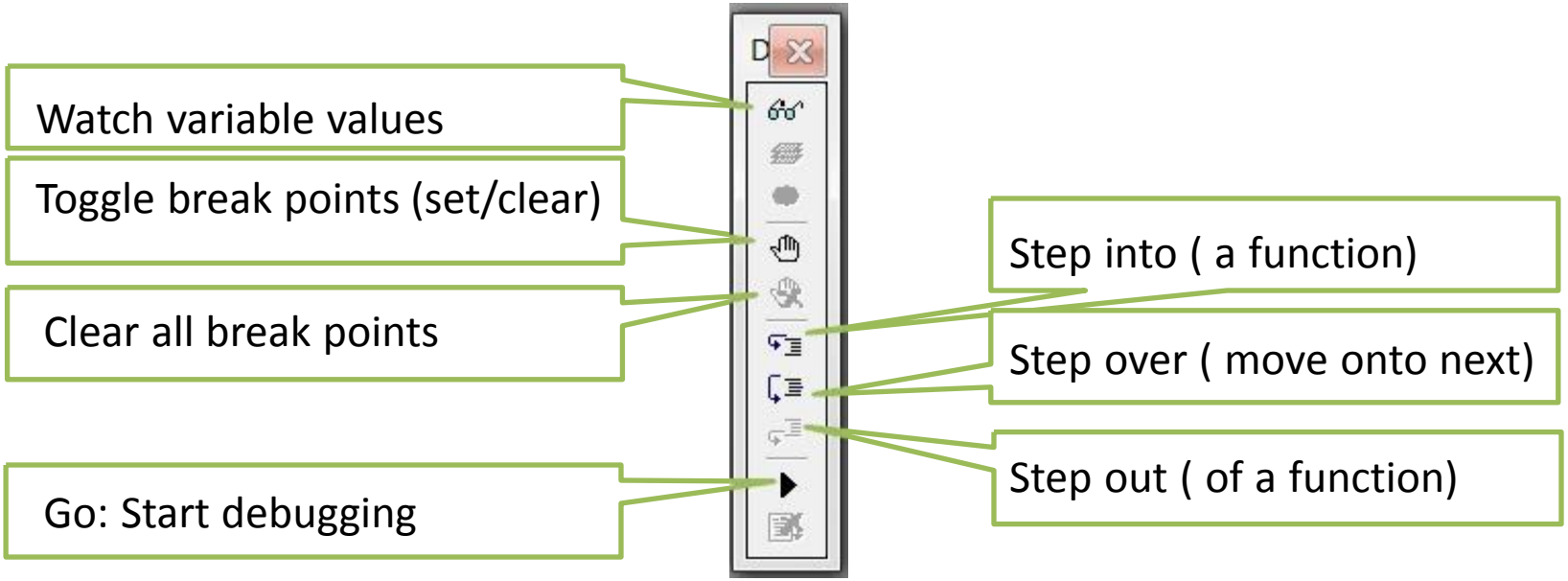
PythonWin Work Environment

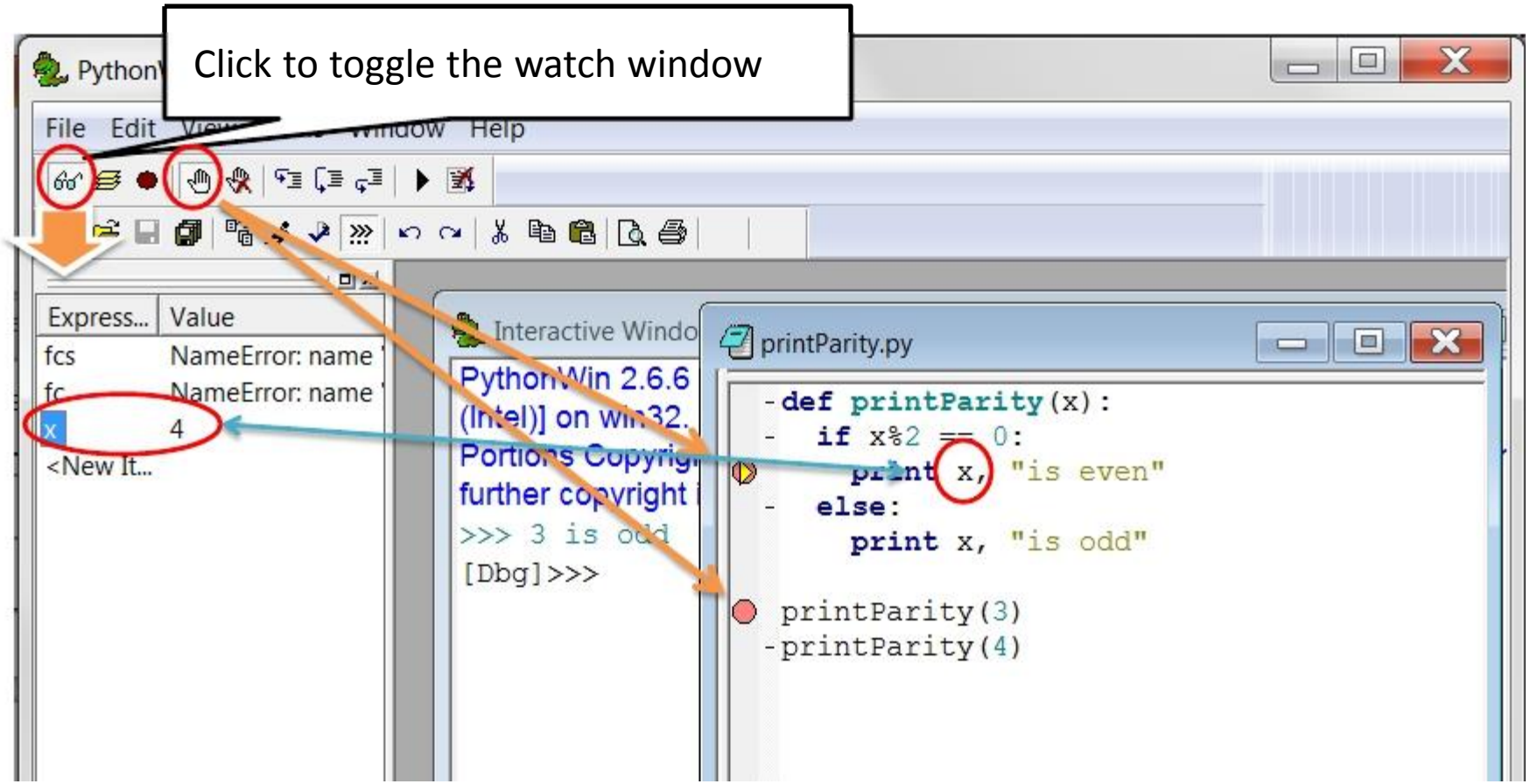


Some quick access tools

Debugging tools



Debugging a program



3 ways to run a Python script in PythonWin

- Type in command in the Interactive window (command line)

```
>>> print 1+2
```

- Run a program file

  a) File -> Run, or
  b) Ctrl + r, or
  c) Click the 'Run' icon  in the tool bar

- Debug a program file

  a) File -> Debug ->Go, or
  b) F5, or
  c) Click the 'Go' icon  in the tool bar

IDCE 302: Chapter 3

Function

Jie Tian, PhD

Clark University

Outline

- Functions
  - Built-in functions
  - Defining new functions
- Execution Flow (multiple functions)
- Main function

What is Function?

- A function is a **<u>named</u> sequence of statements** that performs a computation.
- Defining a function
  - Function name
  - Sequence of statements
- Once properly defined, you can call the function.

Built-in Functions

Type Function

```
>>> type(3.14)
<type 'float'>

>>> type("32")
<type “str“>

>>> type("d:\\data\\landcover.shp")
<type “str“>
```

Type Conversion

```
>>> int("32")  # convert string “32” into an integer 32
32

>>> int("Hello") # converting string “Hello” into an integer?

ValueError: invalid literal for int() with base 10: 'hello'

>>> int(3.99999) # converting 3.99999 into integer, decimals will be truncated.
3

>>> int(-2.3) # converting -2.3 into an integer, decimals will be truncated.
-2

>>> float(32) # converting an integer into a float
32.0

>>> str(32) # converting an integer into a string
"32"

>>> str(3.14149) # converting a float into a string
"3.14149"
```

```
>>> 1/3  # output is an integer
0
>>> 1.0/3  # output is a float
0.3333333333333333

>>> minute = 59
>>> minute / 60.0
0.983333333333

>>> minute = 59
>>> float(minute) / 60   # minute is converted into a float number
0.983333333333
```

Math Functions

- Use site-packages (or modules)
- Need to import
- Use the dot operator to access members (functions and values).

```
>>> import math # you need to import math site package first
>>> math.sqrt(2) / 2.0
0.7071067811865476

>>> math.sin(math.pi/2.0)
1.0

>>> math.radians(180)
3.1415926535897931

>>> math.degrees(math.pi/2)
90.0
```

```
>>> import math as M
>>> M.sqrt(2)/2.0
0.707106781187
```

Some constants in math module

```
>>> math.pi
3.141592653589793

>>> math.e
2.718281828459045

>>> math.log(math.e)
>>> math.log10(10)
```

More Math Functions

```
>>> math.pow(2, 3) # or 2**3
8.0

>>> math.floor(2.9)
2.0

>>> math.ceil(3.0001)
4.0
```

Formatting output precision

```
>>> val = 2.0/3.0
>>> print "%.2f" % val
6.67

>>> print "%10.2f" % val
      6.67
```

6 Spaces

Note: *You are going to see more on this in Chapter 11.*

Defining New Functions

Syntax:

Indentation for the body of a function:

```
def NAME (LIST OF PARAMETERS):
    STATEMENTS
```

Example:

Function definition:

```
# function definition
def printInfo():
    print 'Python version is 2.7!'
```

Function call

```
# Call function
printInfo()
```

```
def calcArea(r):
    area=math.pi*(r**2.0)
    print "Given r= "+ str(r)+", " + \
          "area is : " + str(area)
```

Backslash to split a long line into multiple lines.

Indentation for the body of a function.

```
# function calls
calcArea(1.0)
calcArea(2.0)
```

Execution Flow

```
def WorkdayHours( ):
    print 'Mon-Fri: 9-9'

def WeekendHours( ):
    print 'Weekends: 11-

# call two previously defined functions
def Hours():
    WorkdayHours()
    WeekendHours()

# a function call
Hours()
```

1

2

3

The “main” function?

In C++

```
#include "stdafx.h"
#include <iostream>

int main(){
    std::cout<<"Hello world!";
    return 0;
}
```

In Java

```
public class HelloWorld {
   public static void main(String[] args) {
       System.out.println("Hello world");
    }
}
```

In Python, the main function is not visible:

```
print “Hello World!”
```

Function Arguments

```
import math

# function definition with parameters

def parrot(val): # Val is an argument(or parameter)
    print val

# pass an argument into function call
parrot('I like the color of autumn!')

parrot('I miss the summer beach!')

parrot('Square root of 2 is ' + str(math.sqrt(2.0)))

parrot(math.pow(2, 8))
```

Domain of Variables

- Arguments are **local** variables within the function.

```
 # function definition with parameters
def parrot(val): # Val is an argument(or parameter)
  print val

# pass an argument into function call
parrot('I like the color of autumn!')

print val
```

```
Traceback (most recent call last):
  File "<interactive input>", line 1, in <module>
NameError: name 'val' is not defined
```

Functions with Results

- Some functions need a returned value!
- Need to write **return** statement in these functions.

```
# area calculation
import math

def area(radius):
    return math.pi * radius**2


def absoluteValue(x):
    if x < 0:
        return -x
    else:
        return x
```

Summary

- The standard library has many functions.
- Import modules (site-packages or libraries)
- Access functions or values of modules (dot operator)
- Functions can call each other (be clear with execution flow)
- Define new functions (with or without arguments, require or do not require value(s) returned)

In-class Exercise

- Write a function to calculate the area of a circle.
- The function takes radius as its input parameter, and returns the area.
- Print the calculated area by calling the function.

```
"""
Input:
   r, radius of a circle
Return:
   area of the circle
"""
import math
def CircleArea(r):
    [redacted]
# function call
print CircleArea(1.0)  # call the function with a given radius
print CircleArea(2.0)
```

- Define an happyBday() function
- Call the function with different input strings

```
"""
Input:
   friend's name
Print:
   Happy Birthday to you!
   Happy Birthday to you!
   Happy Birthday, dear <friend's name>.
   Happy Birthday to you!

"""
```