INTERMEDIATE TOPICS, CLI AND EXCEPTIONS

CHRISTOPH PAULIK

DEBUGGING

I will use Spyder and the command line but should be very similar in most IDE's.

```
Open your editor. It might have Python debugging support (Pycharm, P
ydev, Spyder, Emacs ...)
```

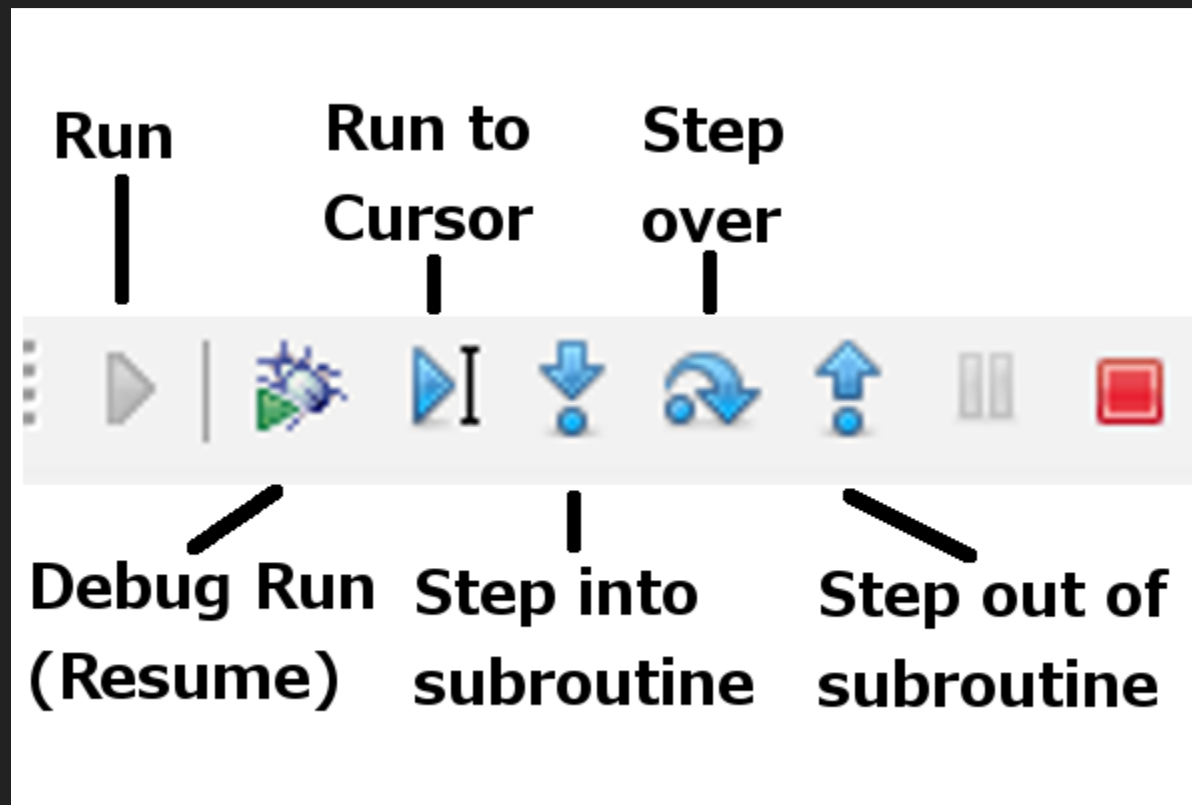# DEBUGGING ACTIONS

Breakpoint
  clicking left of line in most IDE's

Figure 1: Main Debugging Buttons in PyScripter

INTERACTING WITH THE DEBUGGER

Python Interpreter
   write code you want to test
Hover over variable
   shows its value
Variables
   list of all defined variables
Watches
   Watch variables

WHAT IF WE DO NOT HAVE A GUI

The following code will set a breakpoint:

```python
import pdb # imports the python debugger
pdb.set_trace() # sets a breakpoint
# does also work with ipython
# import ipdb
# ipdb.set_trace()
```

Start a python script in debug mode:

```
python -m pdb file.py
```

```python
# Debugging
var1 = "test"
var2 = [1, 2, 3, 4]
var3 = {"key1": 1, "key2": 2}
print(var1, var2, var3) # set a breakpoint here
var1 = "modified"
var3["key5"] = "a new value"
```

```
('test', [1, 2, 3, 4], {'key2': 2, 'key1': 1})
```

LIST AND DICT COMPREHENSION

- for creation of lists or dictionaries based on some loop
- shorter than a classic for loop

```python
dataset1 = [1.73, 80, 2.4122, -4]
threshold = 2.
result = [x > threshold for x in dataset1]
print(result)
```

```
>>> >>> [False, True, True, False]
```

DICTIONARIES

- we can also construct a dictionary

```python
cl = {True: 'larger', False: 'smaller'}
result = {k: cl[k>threshold] for k in dataset1}
print result
```

```
>>> {80: 'larger', 1.73: 'smaller', -4: 'smaller', 2.4122: 'larger'}
```

MULTIPLE ARGUMENTS

```python
data = ['a', 'b', 'c', 'd']
result = {i: x for i,x in enumerate(data)}
print result
```

```
{0: 'a', 1: 'b', 2: 'c', 3: 'd'}
```

~~MINI~~ TINY EXERCISE

```
################################################################
# Tiny Exercise

# Create a list that contains the square value of every
# element in dataset1 using list comprehension.
################################################################
```

SOLUTION

```python
square = [x**2 for x in dataset1]
print(square)
```

```
[2.9929, 6400, 5.818708839999999, 16]
```

ADVANCED FUNCTION ARGUMENTS

Arguments to functions given as lists or tuples can be unpacked by Python

```python
def multi(a, b):
    """Documentation does not fit on slide"""
    return a * b

print(multi(2,3))
numbers = [3, 4]
print(multi(*numbers))
```

```
... ... >>> 6
>>> 12
```

The * tells Python to unpack the arguments

```
print multi(numbers)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: multi() takes exactly 2 arguments (1 given)
```

KEYWORDS

```python
def multikw(arg1="dummy", arg2="text"):
    """Documentation does not fit on slide"""
    return " ".join([arg1, arg2])

kwargs = {'arg1': "this text comes from",
          'arg2': "a keyword dictionary"}

print(multikw())
print(multikw(**kwargs))
```

```
dummy text
this text comes from a keyword dictionary
```

* AND ** IN FUNCTION DEFINITIONS

* will be a tuple, ** a dictionary

```python
def multi(*args):
    """
    Multiplies all given numbers
    """
    print(type(args))
    print("{} Arguments to multiply".format(len(args)))
    res = 1
    for arg in args:
        res *= arg
    return res
print(multi(2,3,4,5,6))
```

```
<type 'tuple'>
5 Arguments to multiply
720
```

**

```python
def print_kw(**kwargs):
    """print keywords"""

    print(type(kwargs))
    for key in kwargs:
        print("{}: {}".format(key, kwargs[key]))

print_kw(argument1=45, argument2="string", test="hello")
```

```
<type 'dict'>
test: hello
argument2: string
argument1: 45
```

FUNCTIONS EVERYWHERE

- Every object can be passed into a function
- e.g. another function

```python
def do(f, a, b):
    print f.__doc__  # this is the docstring of the function
    return f(a,b)
def add(a, b):
    """addition"""
    return a+b
def sub(a, b):
    """subtraction"""
    return a-b

print(do(add, 2, 3))
print(do(sub, 3, 2))
```

```
addition
5
subtraction
1
```

EXCEPTIONS

Are raised when something goes wrong. But can be catched/excepted.

```python
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print "division by zero!"
    else:
        print "result is", result
    finally:
        print "executing finally clause"
divide(2, 1)
divide(2, 0)
```

```
result is 2
executing finally clause
division by zero!
executing finally clause
```

```
print(divide("2", "1"))
```

```
Traceback (most recent call last):
  File "<stdin>", line 12, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

COMMAND LINE INTERFACE (CLI)

ARGUMENTS TO YOUR SCRIPTS

Stored in `sys.argv`

contents of file `cli_1.py`

```python
if __name__ == '__main__':
    import sys
    print sys.argv
```

```
python cli_1.py test -m hello
```

```
['cli_1.py', 'test', '-m', 'hello']
```

ARGPARSE

official tutorial

contents of file `cli_2.py`

```python
import argparse
if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description="get the given name, optionally also the surname
")
    parser.add_argument("given_name", help="given name of the person
")
    parser.add_argument(
        "-s", "--surname", help="the surname of the person")
    args = parser.parse_args()
    print(args.given_name)
    if args.surname:
        print(args.surname)
```

GETTING USER INPUT

contents of file `user_input.py`

```python
num = raw_input("Give me a number: ")
print(type(num))
num = float(num)
print("This is the number you have given me: {:.2f}".format(num))
```

What happens if we do not give a number?

MINI EXERCISE

```
##############################################################
# Mini Exercise

# write a function that makes sure that the input is
# a number
##############################################################
```

SOLUTION

We can fix the user input problem

```python
def get_float():
    while True:
        try:
            num = float(raw_input("Give me a number: "))
            break
        except ValueError:
            print "Oops!  That was no valid number.  Try again..."
    return num

num = get_float()
print("This is the number you have given me: {:.2f}".format(num))
```