Conditionals and Recursion

Chapter 5

Floor Division

- Floor division operator (//) returns an integer dropping the fraction part

```
>>> minutes = 105
>>> minutes / 60
1.75

>>> hours = minutes // 60
>>> hours
1
```

Modulus

- Modulus operator (%) divides two numbers and returns the remainder

Ex:

```
>>> remainder = minutes - hours * 60
>>> remainder
45

>>> remainder = minutes % 60
>>> remainder
45
```

Some Uses of Modulus

- Divisibility: to check if one number is divisible by another
  Ex:
  if `x % y` is zero

- Digit extraction: to get the right-most digit(s) of a number
  Ex:
  ```
  x % 10
  x % 100
  ```
  ...

Boolean Expressions

- An expression that is either true or false.

```
>>> 5 == 5
True
>>> 5 == 6
False
```

- Compares and produces *True* or *False* which is a special type called **bool**.

```
>>> type(True)
<class ' bool' >
>>> type(False)
<class ' bool' >
```

Relational Operators

```
x == y     # x is equal to y
x != y     # x is not equal to y

x > y      # x is greater than y
x < y      # x is less than y

x >= y     # x is greater than or equal to y
x <= y     # x is less than or equal to y
```

Relational Operators (cont.)

```
x = 3
y = 4

print (x = y)

print(x =< y)
```

Logical Operators

- `and, or, not`

Ex:
We will go to picnic:

- if the sun is shiny **and** the weather is good
- if we find a car **or** a bus
- if it is **not** rainy

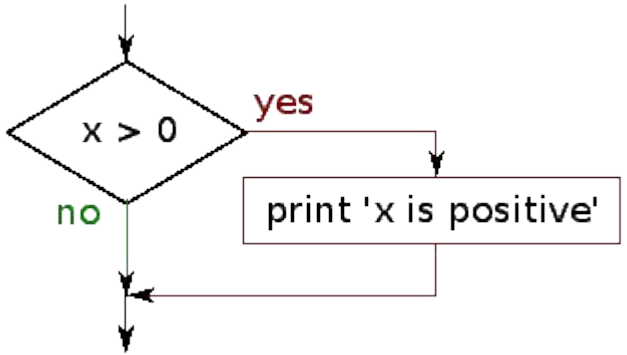
```
x > 0 and x < 10          # true if both are true
n%2 == 0 or n%3 == 0      # true if either or both are true
not (x > y)               # true if false
```

Logical Operators (cont.)

```
>>> x = 13
>>> x>0 and x<10     # operands should be boolean expressions
>>> x>0 or x<10

>>> True or False

>>> 13 and True
>>> True and 13

>>> 13 or Iamsoconfused
>>> Iamsoconfused or 13
```

Conditional Execution

```
if x > 0:
    print(' x is positive' )
```



- **Conditional statement**
- Has the same structure as function definition.
- **Condition:** boolean expression that is after **`if`**.
- At least one statement in the body or:

```
if x < 0:
    pass   # TODO: need to handle negative values!
```

Alternative Execution

```
if x % 2 == 0:
    print(' x is even' )
else:
    print(' x is odd' )
```

Chained Conditionals

```
if x < y:
    print(' x is less than y' )
elif x > y:
    print(' x is greater than y' )
else:
    print(' x and y are equal' )
```

- **`else`** is an abbreviation of “else if”
- Exactly one branch runs
- No limitation on the number of **`elif`** statements
- If there is an **`else`** clause, it has to be at the end
- Each condition is checked in order

Chained Conditionals (cont.)

```
x = 2
y = 3

if x < y:
    print(' x is less than y' )
if x > y:
    print(' x is greater than y' )
if x == y:
    print(' x and y are equal' )
```

Chained Conditionals (cont.2)

```
x = 3

if x > 0:
    print(' x is positive' )
elif x % 2:       # ?
    print(' x is odd' )
elif type(x) == int:
    print(' x is an integer' )
```

Nested Conditionals

```
if x == y:
    print(' x and y are equal' )
else:
    if x < y:
        print(' x is less than y' )
    else:
        print(' x is greater than y' )
```

Nested Conditionals (cont.)

- Difficult to read very quickly, thus, if possible avoid.

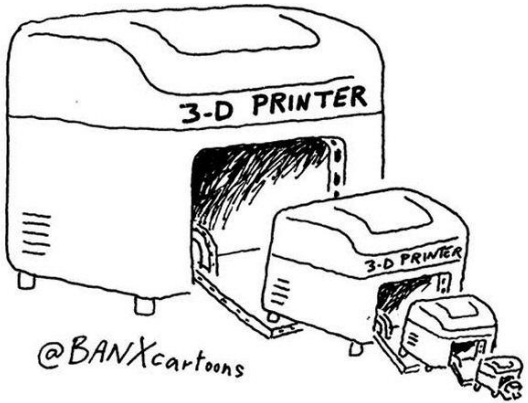
```
If Number = 1 Then
    Count1 = Count1 + 1
Else
    If Number = 2 Then
        Count2 = Count2 + 1
    Else
        If Number = 3 then
            Count3 = Count3 + 1
        Else
            CountX = CountX + 1
        End if
    End If
End If
```

Nested Conditionals (cont.2)

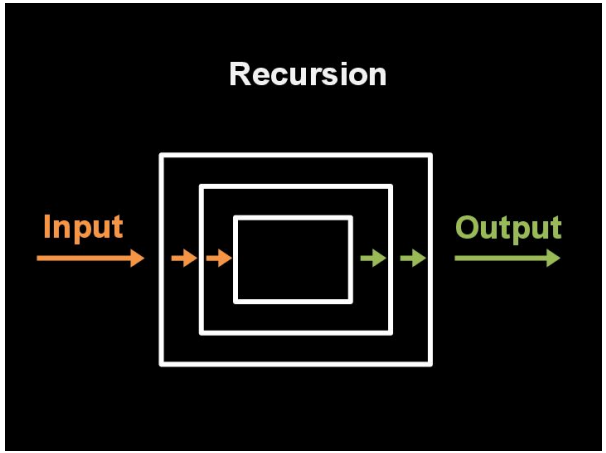
```
if 0 < x:
    if x < 10:
        print(' x is a positive single-digit number. ' )

if 0 < x and x < 10:
    print(' x is a positive single-digit number. ' )


if 0 < x < 10:
    print(' x is a positive single-digit number. ' )
```

Recursion

- A function that calls itself is **recursive**;
  the process of executing it is called **recursion**.




```
def f1():
    print('f1 executed')

def f2():
    print('f2 executed')
    f1()

def f3():
    print('f3 executed')
    f3()
```



Recursion (cont.)

```
def countdown(n):
    if n <= 0:
        print(' Blastoff! ' )
    else:
        print(n)
        countdown(n-1)

>>> countdown(3)
```

- **Base case**

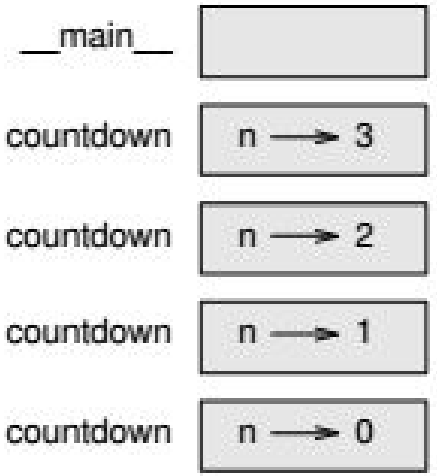
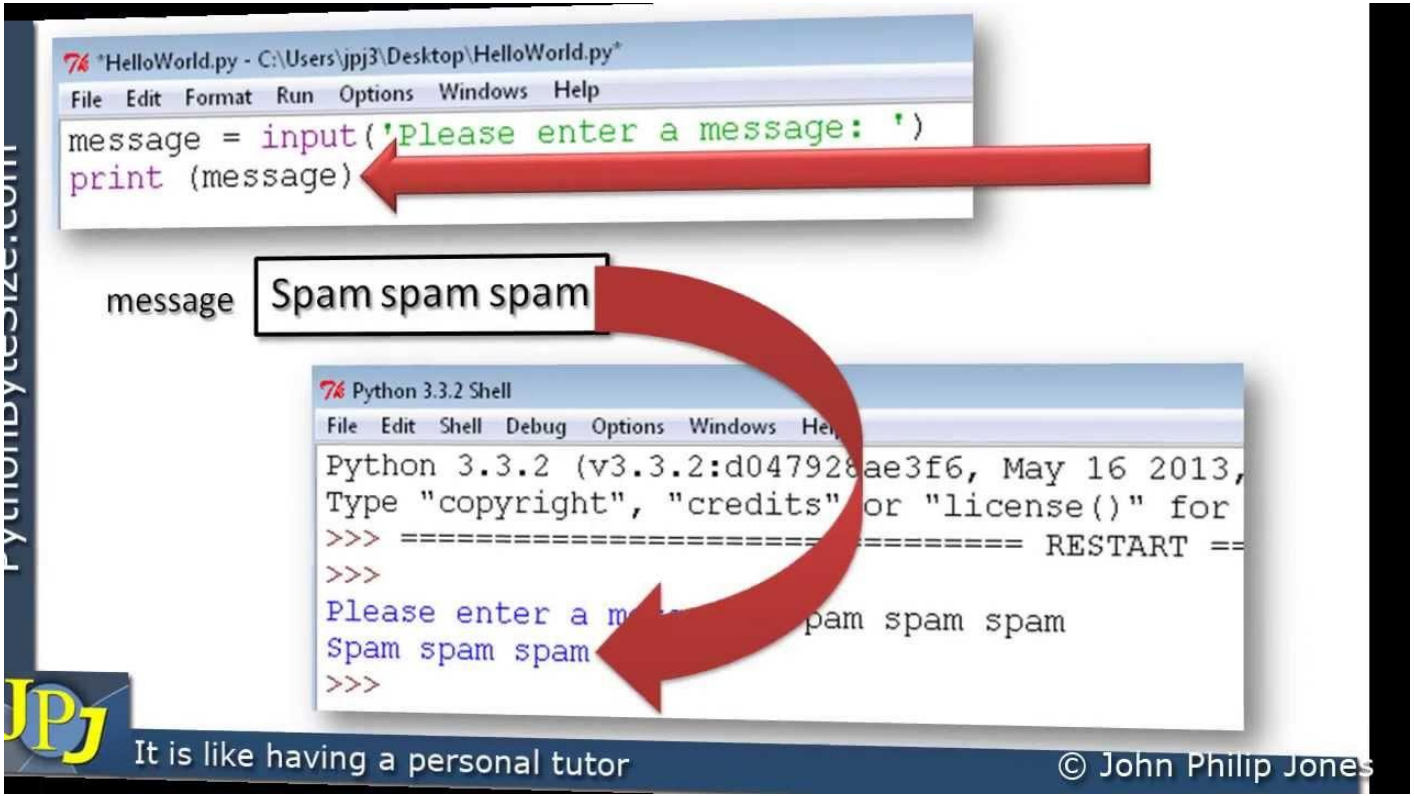


Figure 5.1: Stack diagram.

Keyboard Input



Debugging

- Debugging and Glossary sections are important

Error Types:

- *Syntax error*: related to structure of a program and its rules (before execution)

- *Runtime error*: indicates that something exceptional (bad) has happened (during execution)

- *Semantic error*: related to meaning. No errors, but you will NOT get what you want.