Mathematical Functions – Built In

- Python supports some mathematical functions built in
- No import of a library required

TABLE 3.1 Simple Python Built-in Functions

Function	*Description*	*Example*
`abs(x)`	Returns the absolute value for `x`.	`abs(-2)` is `2`
`max(x1, x2, ...)`	Returns the largest among `x1`, `x2`, ...	`max(1, 5, 2)` is `5`
`min(x1, x2, ...)`	Returns the smallest among `x1`, `x2`, ...	`min(1, 5, 2)` is `1`
`pow(a, b)`	Returns a^b. Same as `a ** b`.	`pow(2, 3)` is `8`

`round(x)`	Returns an integer nearest to `x`. If `x` is equally close to two integers, the even one is returned.	`round(5.4)` is `5`
		`round(5.5)` is `6`
		`round(4.5)` is `4`

`round(x, n)`	Returns the float value rounded to `n` digits after the decimal point.	`round(5.466, 2)` is `5.47` `round(5.463, 2)` is `5.46`

Mathematical Functions - Library

- Python supports even more mathematical functions but require a library import
- Use 'import math'
- Pi and e are defined
  - Use math.pi and math.e

List of Mathematical Functions

- Imported math functions

TABLE 3.2 Mathematical Functions

Function	*Description*	*Example*
`fabs(x)`	Returns the absolute value for `x` as a float.	`fabs(-2)` is `2.0`
`ceil(x)`	Rounds `x` up to its nearest integer and returns that integer.	`ceil(2.1)` is `3` `ceil(-2.1)` is `-2`
`floor(x)`	Rounds `x` down to its nearest integer and returns that integer.	`floor(2.1)` is `2` `floor(-2.1)` is `-3`
`exp(x)`	Returns the exponential function of `x` (e^x).	`exp(1)` is `2.71828`
`log(x)`	Returns the natural logarithm of `x`.	`log(2.71828)` is `1.0`
`log(x, base)`	Returns the logarithm of `x` for the specified base.	`log(100, 10)` is `2.0`
`sqrt(x)`	Returns the square root of `x`.	`sqrt(4.0)` is `2`
`sin(x)`	Returns the sine of `x`. `x` represents an angle in radians.	`sin(3.14159 / 2)` is `1` `sin(3.14159)` is `0`
`asin(x)`	Returns the angle in radians for the inverse of sine.	`asin(1.0)` is `1.57` `asin(0.5)` is `0.523599`
`cos(x)`	Returns the cosine of `x`. `x` represents an angle in radians.	`cos(3.14159 / 2)` is `0` `cos(3.14159)` is `-1`
`acos(x)`	Returns the angle in radians for the inverse of cosine.	`acos(1.0)` is `0` `acos(0.5)` is `1.0472`
`tan(x)`	Returns the tangent of `x`. `x` represents an angle in radians.	`tan(3.14159 / 4)` is `1` `tan(0.0)` is `0`
`degrees(x)`	Converts angle `x` from radians to degrees.	`degrees(1.57)` is `90`
`radians(x)`	Converts angle `x` from degrees to radians.	`radians(90)` is `1.57`

STRINGS

TABLE 3.3 Python Escape Sequences

Character Escape Sequence	*Name*	*Numeric Value*
`\b`	Backspace	8
`\t`	Tab	9
`\n`	Linefeed	10
`\f`	Formfeed	12
`\r`	Carriage Return	13
`\\`	Backslash	92
`\'`	Single Quote	39
`\"`	Double Quote	34

Boolean Types

- How do we compare values in Python
- We use comparison operators (6 different ones available in Python)

Python Operator	Mathematics	Name	Example (radius is 5)	Result
<	<	Less than	radius < 0	False
<=	≤	Less than or equal to	radius <= 0	False
>	>	Greater Than	radius > 0	True
>=	≥	Greater than or equal to	radius >= 0	True
==	=	Equal to	radius == 0	False
!=	≠	Not equal to	radius != 0	True

Boolean Types Examples

- radius = 1
- print(radius > 0) # True
- lightsOn = True # Assign True to lightOn
- Result of printing True or False is either 1 or 0
- print(int(True)) # Prints 1
- print(int(False)) # Prints 0

bool Function

- Bool function converts a numeric value to a Boolean value
- Returns False if the value is 0; otherwise if always returns True
- print(bool(0)) # Prints False
- print(bool(4)) # Prints True
- print(bool(5)) # Prints True

If Statement Structure

Check a condition

colon

```
if condition:
    #if the check passes do this code
```

Indent

If statement passes, then execute this code

If-Else Statement

Check a condition

```
if condition:
    # Run this code block
else:
    # If not the statement, always run
    this statement
```

If not the statement – ***ALWAYS*** run the else block

Elif Statement Structure

Check a condition

```
if statement:
        # Run this code block
elif statement:
        # Run this code clock
else:
        # If none of the above statements,
        always run this statement
```

Check another condition. (*n* checks)

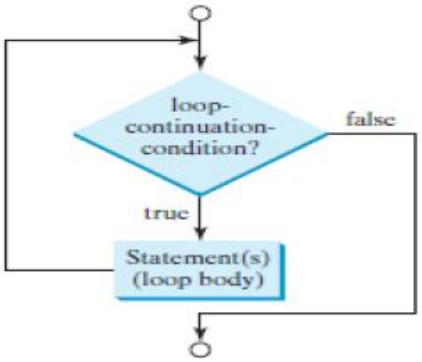
If none of the statements – ***ALWAYS*** run the else block

Program: Hello World Repeated

```
count = 0
while count < 100:
        print("Hello World")
        count = count + 1
```

This prints Hello World 100 times

While Loop Flow Chart



(a) A while loop

Controlling a Loop with User Confirmation

```
continueLoop = 'Y'
while continueLoop == 'Y':
        #Execute the loop body once
        ...

        #Prompt the user for confirmation
        continueLoop = input("Enter Y to continue and N to quit: ")
```

For Loops

- Similar to while loops
- Used when you know exactly how many times the loop body needs to run
- Called a counter-controlled loop

```
i = initialValue
while i < endValue:
        #Loop Body
        i += 10

for i in range(initialValue, endValue):
        #Loop Body
```

For Loop Example

```
for v in range(4, 8):
        print(v)
```

Prints 4, 5, 6, 7

Range (a, b, k) Example

```
for v in range(3, 9, 2):
        print(v)
```

Prints 3, 5, 7

Range Function

- range(a) and range(a, b)
- range(a)
  - range(0, a)
- range(a, b)
  - range(2, 5)
- range(a, b, k)
  - k is the step
  - Start at a, step by k, end at b

Nested Loops

- Consists of an outer loop and one or more inner loops
- Each time the outer loop is repeated, the inner loops are reentered and started new

```
for i in range(1, 10):
        print("Hello 1")
        for j in range(1, 10):
                print("Hello 2"")
```

Notice Nested Loops

```
for i in range(1000):
        for j in range(1000):
                for k in range(1000):
                        #Do Something
```

- This loop is ran 1,000,000 times!
- Assuming each iteration takes 1 millisecond, this would run 277 hours

Break – Additional Loop Control

- Immediately terminates a loop
- As soon as a break is encountered we leave the loop

```
sum = 0
number = 0

while number < 20:
    number += 1
    sum += number
    if sum >= 100:
        break

print("The number is", number)
print("The sum is", sum)
```

Break example – Broken Down

```
sum = 0
number = 0

while number < 20:
    number += 1
    sum += number
    if sum >= 100:
        break

print("The number is", number)
print("The sum is", sum)
```

Break statement – we leave the loop

For Continue – Additional Loop Control

- End the current iteration and jump to the end of the loop body
- Breaks out of an iteration

```
sum = 0
number = 0

while number < 20:
    number += 1
    if number == 10 or number == 11:
        continue
    sum += number

print("The sum is", sum)
```

Break vs. Continue

Break

- Terminates loop
- Leave loop body immediately

Continue

- End current iteration
- Jumps to end of loop body
- Loop does not end

Functions

- A way to define ***reusable*** code and ***organize*** and ***simplify*** code

- What happens when we need to draw 8 squares on screen?
- We have to type in that code line after line
- Functions allow us to reuse code
- () mean function. print() <- parenthesis here mean function

Basic Outline for Programs with Functions

```
def functionName():
        # Function Body

def main():
        # Call functions here and do other things

main()
```

Function with Parameters

```
def functionName(list of parameters):
        # Function Body
```

Example: Funtion with Parameters

```
def sum(number1, number2):
        answer = number1 + number2
        print(answer)

sum(2, 2)
```

Function without Parameters

```
def functionName():
        # Function Body
```

Function with/without Return Values

With return value

```
def functionName():
        # Function Body
        return someValue

def main():
        variable = functionName()
```

Without return value

```
def functionName():
        # Function Body

def main():
        functionName()
```

Example: Finding the Max Number

```
def max(number1, number2):
        if number1 > number2:
                result = number1
        else:
                result = number2

        return result

def main():
        answer = max(5 , 2)
        print(answer)
```

Python 3's Print Function Source

```
def print_(*args, **kwargs):
    """The new-style print function from py3k."""
    fp = kwargs.pop("file", sys.stdout)
    if fp is None:
        return
    def write(data):
        if not isinstance(data, basestring):
            data = str(data)
        fp.write(data)
    want_unicode = False
    sep = kwargs.pop("sep", None)
    if sep is not None:
        if isinstance(sep, unicode):
            want_unicode = True
        elif not isinstance(sep, str):
            raise TypeError("sep must be None or a string")
    end = kwargs.pop("end", None)
    if end is not None:
        if isinstance(end, unicode):
            want_unicode = True
        elif not isinstance(end, str):
            raise TypeError("end must be None or a string")
    if kwargs:
        raise TypeError("invalid keyword arguments to print()")
    if not want_unicode:
        for arg in args:
            if isinstance(arg, unicode):
                want_unicode = True
                break
    if want_unicode:
        newline = u"\n"
        space = u" "
    else:
        newline = "\n"
        space = " "
    if sep is None:
        sep = space
    if end is None:
        end = newline
    for i, arg in enumerate(args):
        if i:
            write(sep)
        write(arg)
    write(end)
```