Week 1

Video 2: Getting Started: Installing Python

Programming Languages

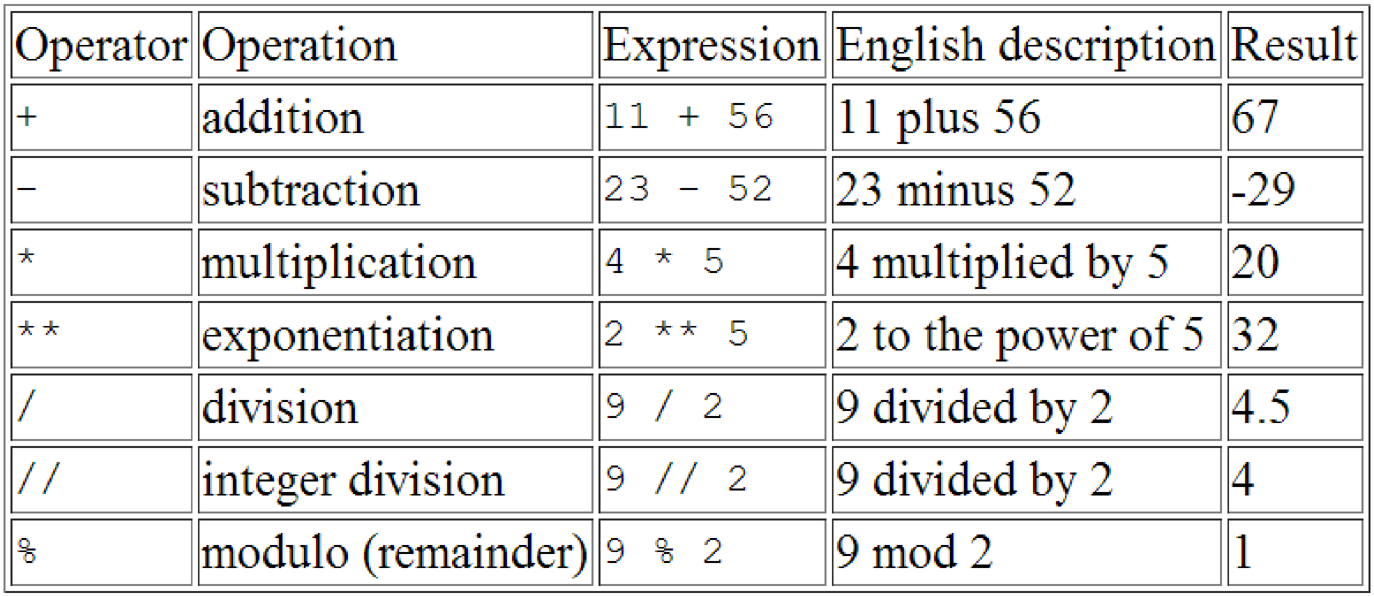
Programs are found in many places such as on your computer, your cell phone, or on the internet. A program is a set of instructions that are run or executed. There are many programming languages and, for this course, we will use the Python programming language.

Optional Book Reading

Chapter 1: What's Programming?
Chapter 2: How does a Computer Run a Python Program?

Video 3: Python as a Calculator

Arithmetic Operators



Operator	Operation	Expression	English description	Result
`+`	addition	`11 + 56`	11 plus 56	67
`-`	subtraction	`23 - 52`	23 minus 52	-29
`*`	multiplication	`4 * 5`	4 multiplied by 5	20
`**`	exponentiation	`2 ** 5`	2 to the power of 5	32
`/`	division	`9 / 2`	9 divided by 2	4.5
`//`	integer division	`9 // 2`	9 divided by 2	4
`%`	modulo (remainder)	`9 % 2`	9 mod 2	1

Types *int* and *float*

A *type* is a set of values and operations that can be performed on those values.

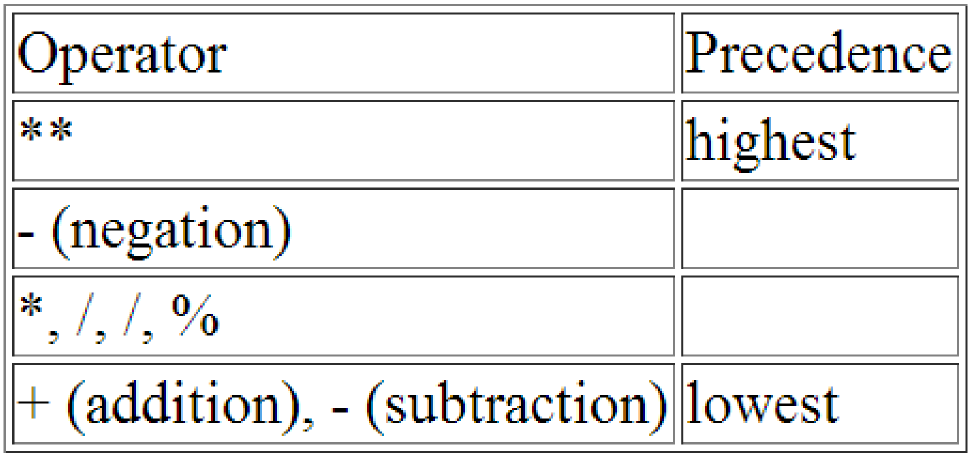
Two of Python's numeric types:

- int: integer
  For example: 3, 4, 894, 0, -3, -18

- float: floating point number (an approximation to a real number)
  For example: 5.6, 7.342, 53452.0, 0.0, -89.34, -9.5

Arithmetic Operator Precedence

When multiple operators are combined in a single expression, the operations are evaluated in order of precedence.



Operator	Precedence
**	highest
- (negation)	
*, /, /, %	
+ (addition), - (subtraction)	lowest

Syntax and Semantics

Syntax: the rules that describe valid combinations of Python symbols

Semantics: the meaning of a combination of Python symbols is the meaning of an instruction â€" what a particular combination of symbols does when you execute it.

Errors

A syntax error occurs when we an instruction with invalid syntax is executed. For example:

```
>>> 3) + 2 * 4
SyntaxError: invalid syntax
```

A semantic error occurs when an instruction with invalid semantics is executed. For example:

```
>>> 89.4 / 0
Traceback (most recent call last):
    File "", line 1, in
        89.4 / 0
ZeroDivisionError: float division by zero
```

Video 4: Python and Computer Memory

Computer Memory

For the purpose of this course, you may think of *computer memory* as a long list of storage locations where each location is identified with a unique number and each location houses a value. This unique number is called a memory address. Typically, we will write memory addresses as a number with an "id" as a prefix to distinguish them from other numbers (for example, **id201** is memory address 201).

Variables are a way to keep track of values stored in computer memory. A variable is a named location in computer memory. Python keeps variables in a separate list from values. A variable will contain a memory address, and that memory address contains the value. The variable then refers to the value. Python will pick the memory addresses for you.

Terminology

A value **has** a memory address.
A variable **contains** a memory address.
A variable **refers** to a value.
A variable **points** to a value.

Example:
Value 8.5 **has memory address** id34.
Variable shoe*size* ***contains memory address*** *id34.*
The ***value*** *of shoe*size **is** 8.5.
shoe*size* ***refers*** *to value 8.5.*
*shoe*size **points** to value 8.5.

Optional Book Reading

Chapter 2.4: Variables and Computer Memory: Remembering Values

Video 5: Variables

Assignment statements

The general form of an assignment statement:

```
variable = expression
```

Example assignment statements:

```
>>> base = 20
>>> height = 12
>>> area = base * height / 2
>>> area
120.0
```

The rules for executing an assignment statement:

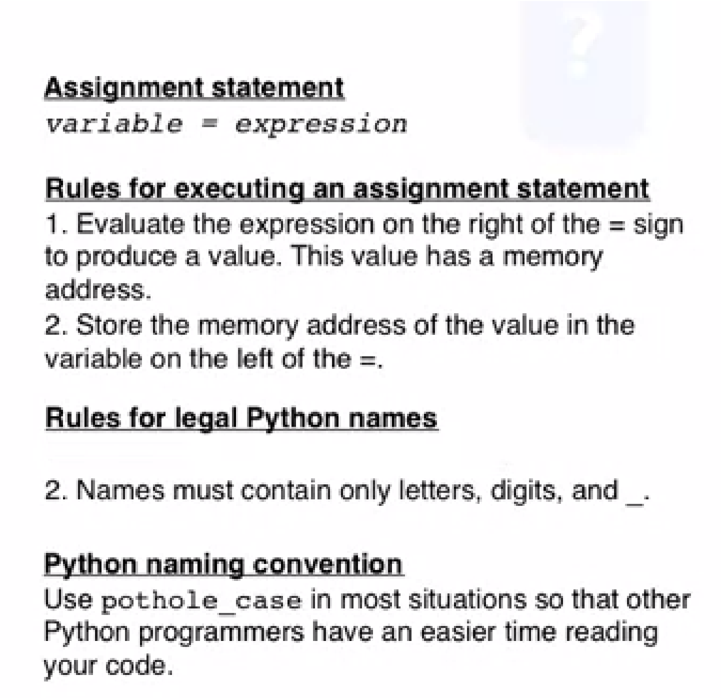
1. Evaluate the expression. This produces a memory address.
2. Store the memory address in the variable.

Variable names

The rules for legal Python names:

1. Names must start with a letter or _.
2. Names must contain only letters, digits, and _.

For Python, in most situations, the convention is to use pothole_case.



Assignment statement

variable = expression

Rules for executing an assignment statement

1. Evaluate the expression on the right of the = sign to produce a value. This value has a memory address.
2. Store the memory address of the value in the variable on the left of the =.

Rules for legal Python names

2. Names must contain only letters, digits, and _.

Python naming convention

Use `pothole_case` in most situations so that other Python programmers have an easier time reading your code.

Optional Book Reading

Chaper 2.4. Variables and Computer Memory: Remembering Values
Chapter 2.6. A Single Statement That Spans Multiple Lines

Video 6: Built-in Functions

Function Call

The general form of a function call:

```
function_name(arguments)
```

The rules for executing a function call:

1. Evaluate the arguments.
2. Call the function, passing in the argument values.

Terminology:

- Argument: a value given to a function
- Pass: to provide to a function
- Call: ask Python to evaluate a function
- Return: pass back a value

Example function calls:

```
>>> abs(-23)
23
>>> abs(56.24)
56.24
```

Function *dir*

Python has a set of built-in functions. To see the list of built-in functions, run dir(__builtins__):

>>> dir(__builtins__)< br/> ['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError', 'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '*', '__build*class__', '__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

Function help

To get information about a particular function, call help and pass the function as the argument. For example:

```
>>> help(abs)
Help on built-in function abs in module builtins:
abs(...)
    abs(number) -> number

    Return the absolute value of the argument.
```

Optional arguments

In the description of function pow below, the square brackets around [, z] indicate that the third argument is optional:

```
>>> help(pow)
Help on built-in function pow in module builtins:

pow(...)
    pow(x, y[, z]) -> number

    With two arguments, equivalent to x\*\*y.  With three arguments, equivalent to (x\*\*y)
```

Function pow can be called with either two or three arguments:

```
>>> pow(2, 5)
32
>>> pow(2, 5, 3)
2
```

Optional Book Reading

Chaper 3.1. Functions That Python Provides
Chaper 3.2. Memory Addresses: How Python Keeps Track of Values

Video 7: Defining Functions

Function Definitions

The general form of a function definition:

```
def function_name(parameters):
    body
```

- **def**: a keyword indicating a function definition
- function_name: the function name
- parameters:
    - The parameter(s) of the function, 0 or more and are separated by a comma
    - a parameter is a variable whose value will be supplied when the function is called
- body:
    - 1 or more statements, often ending with a return statement

Example of a function definition:

```
def f(x):
    return x ** 2
```

return statement

The general form of a return statement:

```
return expression
```

The rules for executing a **return** statement:

1. Evaluate the expression. This produces a memory address.
2. Pass back that memory address to the caller. Exit the function.

Function Calls

Function calls are expressions and the result can be stored in a variable.

The general form of a function call:

```
function_name(arguments)
```

The rules for executing a function call:

1. Evaluate the arguments to produce memory addresses.
2. Store those memory addresses in the corresponding parameters.
3. Execute the body of the function.

Example of a function definition and function calls:

```
>>> def area(base, height):
return base * height / 2
>>> area(3, 4)
6.0
>>> result = area(10, 7.45)
>>> result
37.25
```

Saving your programs to ".py" files

We usually save our Python programs in ".py" files. A file can contain multiple function definitions and other statements. Before calling a function from a ".py" file in the shell in IDLE, you need to first execute Run -> Run Module, or else the shell will not recognize the function call.

Optional Book Reading

Chaper 3.3. Defining Our Own Functions
Chaper 3.4. Using Local Variables for Temporary Storage
Chaper 3.7. Writing and Running a Program