Computer Programming I

Variables and Expressions

Variables

- Variables are references to locations in memory
- Visualization: Imagine there is a table in memory with your variables
- Example for the variables:

```
a = 5
b = 9
```

- In memory this looks like:

a	5
b	9

Assignment

- In Python we create variables by using the assignment operation using the = symbol (assignment operator)
- The name of the variable is on the left side of the equals sign
- The value of the variable is on the right side of the equals sign

```
>>> year = 2020
>>> course = 'CS 171'
```

- Computations on the right side are completed first, then the value is stored in the variable.

Objects

- An object represents a value and is automatically created by the interpreter when executing a line of code.
- Objects are used to represent everything in a Python program, including integers, strings, functions, lists, etc.
- Each Python object has three defining properties:
  - Value: A value such as "20", "abcdef", 3.55, or 55.
  - Type: The type of the object, such as integer or string.
  - Identity: A unique identifier that describes the object.

Objects

- Value: data associated with the object
- Type: determines the object's supported behavior.
  - An object's type never changes once created.
  - The built-in function **`type()`** prints the type of an object.
- Identity: A unique numeric identifier
  - Refers to the memory address where the object is stored
  - Python provides a built-in function **`id()`** that gives the value of an object's identity
- Example:

```
age = 19
print('age:', age)              #age: 19
print('type: ', type(age))      #type: <class 'int'>
print('id:', id(age))           #id: 1478350176
```

Variable Types

- The variable type tells us what kind of data the variable holds.
- The same operators act differently on different types.

```
a = 7
b = 'cat’
c = 2
print(a * b) #'catcatcatcatcatcatcat'
print (a * c) #14
```

- Multiplication means duplicate for strings.

Types: Numbers

- There are two different numerical types
  - Integer: Whole numbers
  - Float: Decimal numbers
- Examples

```
>>> gpa = 3.55          #float
>>> age = 20            #integer
>>> balance = -45.55    #another float
```

- In general we use:
  - integers to represent values that can be counted
  - floats to represent values that are measured

Floating-point numbers

- A Floating-Point number is a *real number*.
- The term floating-point refers to the decimal point floating to any point in the number.
- float is the Data Type used for Floating Point Numbers
- Floating Point Literals are always written with a decimal, even if it is 0.
- Examples:
  - `1.0` is a float
  - `1.5` is a float

Reading Float numbers

- Use the built-in function **`float()`** reading the input to convert the input string into a float
- Example:

```
>>> gpa = float(input("Enter your gpa: "))
Enter your gpa: 3.75
```

Scientific Notation

- Useful for representing floating-point numbers that are much greater than or much less than 0 (zero)
  - A decimal number with 1 digit before the decimal point
  - A fixed number of decimal digits
  - The **`E`** (or **`e`**) symbol
  - The exponent of 10
- Example:

  `3.25E5` = 3.25 * 10^{5} = 325000

  `1.0e-3` = 1.0 * 10^{-3} = 0.001

Arithmetic Expressions

- Arithmetic Expression is a combination of literals, variables, parentheses, and arithmetic operators.
  - It yields a result
  - literal: a constant value used in the program
  - operator is a symbol that performs a calculation
  - Parenthesis can be used to group operations

Arithmetic Operators

SYMBOL	OPERATION	Description/Notes
+	Addition	Adds two numbers
-	Subtraction	Subtracts one number from another Also use for negation
*	Multiplication	Multiplies two numbers
/	Division	Divides left hand operand by right hand operand
//	Floor Division	Rounds down the result of a division to the closest whole number value.
%	Remainder, or modulo	Divides left hand operand by right hand operand and returns remainder
**	Exponent	Performs exponential (power) calculation on operators

Order of Operations

1. Perform any operations that are enclosed in parentheses.
2. Perform any exponentiation
3. Perform any negation
4. Perform any multiplications, divisions, or modulo operations as they appear from left to right.
5. Perform any additions or subtractions as they appear from left to right.

Order of Operations

Expression	Value
`5 + 2 * 4`	13
`10 / 2 – 3`	2
`8 + 12 * 2 - 4`	28
`6 – 3 * 2 + 7 - 1`	6

Order of Operations

Expression	Value
`(5 + 2) * 4`	28
`10 / (5 - 3)`	5
`8 + 12 * (6 - 2)`	56
`(6 - 3) * (2 + 7) / 3`	9

Expressions and Assignment

- An expression's evaluation does not change memory.
- Typically we want to assign the returned value to a variable.
- Example:

```
x = 10
y = 7
z = x * y
print(z)
```

Combined assignment operators

- Modify the value of a variable and store the result back into the same variable.

Operator	Example Usage	Equivalence
+=	x += 5	x = x + 5
-=	y -= 2	y = y -2
*=	z *= 10	z = z * 10
/=	a /= b	a = a / b
%=	c %= 3	c = c % 3

Division

- The division operator returns a decimal number.
- Example:

```
a = 17 / 9 #Sets a to 1.8888888888888888
```

- We may also want to know the quotient and remainder.
- Example:

```
17 / 9 is also 1 with remainder 8 because
17 = 9 * 1 + 8
```

Division

- To find the quotient of the division we use the floor division operator //

```
q = 17 // 9 # q is 1
```

- To find the remainder of the division we use the modulo operator %

```
r = 17 % 9  #r is 8
```

- The quotient and remainder are always exact
  - No decimal places → integers

Numerical Error

- Floating Point math can introduce rounding errors
- Example:

```
a = 2.56710E50
b = 7
print( a + b )
#Prints 2.567e+50
```

- There are not enough digits to store the sum accurately.

Numerical Error

- Overflow: Values that are too large/small for memory
- Example:

```
>>> print(2.0 ** 1024)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
OverflowError: (34, 'Result too large')
```

Numerical Error

- Floats can introduce errors
- When to use float:
  - Measurements such as distances, temperatures, volumes, etc.
  - Situations where perfect accuracy is not needed.
  - Computing an average.

Bits and Bytes

- Everything a computer does uses bits.
- A bit is the smallest unit of memory in a computer.
- A single bit can have one of two values: 0 or 1.
- A byte is 8 bits.
- Most work on a computer is does in bytes.
- An 8-bit computer (NES/Atari 26000) can work with 1 byte at a time
- A 64-bit computer (modern) can work with 8 bytes at a time
- 128-bit computers have been built for research.

Storing an Integer in Binary

- How is a number like 37 stored in memory using just `0`s and `1`s?
- We assign a value to each bit in a byte.
- Each position has either
  - 0 - This value is not part of the number
  - 1 - This value is part of the number
- Example: 37 = 32 + 4 + 1

Power of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
Part of Num.	0	0	1	0	0	1	0	1

Even Numbers

- Every Even number can be made by adding twos

  $6 = 2 * 3 = 2 + 2 + 2$

  $88 = 2 * 44 = 2 + 2 + 2 + \ldots.$

- We can make any even number by adding powers of 2.

  $6 = 4 + 2 = 2^2 + 2$

  $88 = 64 + 16 + 8 = 2^6 + 2^4 + 2^3$

Odd Numbers

- Any number taken to the 0 exponent is 1.
- Example: $2^0 = 1$
- Any odd number is just an even number plus 1.

$$7 = 6 + 1 = 2^2 + 2^1 + 2^0$$

$$89 = 88 + 1 = 2^6 + 2^4 + 2^3 + 2^0$$

- Every number is either even or odd
- Conclusion: Every positive number can be created by adding powers of 2.

Binary Examples

Value	128	64	32	16	8	4	2	1
91	0	1	0	1	1	0	1	1
139	1	0	0	0	1	0	1	1
14	0	0	0	0	1	1	1	0
200	1	1	0	0	1	0	0	0

Largest Binary Numbers

- What is the largest number we can t in 8-bits?
- Set every bit to be `1`.

```
128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255
```

Value	128	64	32	16	8	4	2	1
255	1	1	1	1	1	1	1	1

Largest Binary Numbers

- 4-bit Computer: $15 = 2^4 - 1$
- 8-bit Computer: $255 = 2^8 - 1$
- 16-bit (Sega/SNES): $65{,}525 = 2^{16} - 1$
- 32-bit (Windows XP): $4{,}294{,}967{,}295 = 2^{32} - 1$
- 64-bit (Modern): $18{,}446{,}744{,}073{,}709{,}551{,}615 = 2^{64} - 1$

Remainder, Division and Binary

- Remainders and floor division are used to convert number from decimal to binary
  - The Remainder: current bit.
  - The Quotient: Remaining work.

Remainder, Division and Binary

```
198 % 2 = 0
 99 % 2 = 1
 49 % 2 = 1
 24 % 2 = 0
 12 % 2 = 0
  6 % 2 = 0
  3 % 2 = 1
  1 % 2 = 1
```

```
198 // 2 = 99
 99 // 2 = 49
 49 // 2 = 24
 24 // 2 = 12
 12 // 2 = 6
  6 // 2 = 3
  3 // 2 = 1
  1 // 2 = 0
```

Binary Value of `198` is `11000110`

Modules

- Frequently we will want to reuse values in multiple programs
- These values can be stored in separate files and used by multiple scripts
- Files used in this way are called Modules
- Python provides many built-in modules for common tasks (Python standard library)
  - For example, the math and the random modules

Module Example

- If we are working on multiple physics programs, it may be helpful to put all the useful constants in a single file.
- Below is some code from a file `constants.py`

```
Gravity = 6.67 * 10 ** (-11)    #in m^2/kg/s^2
EarthMass = 5.97 * 10 ** (24)    #in kg
EarthRadius = 6.378 * 10 ** (6) # in m
```

Modules vs. Scripts

- The code you execute directly in the interpreter is called a script.
- For example, we could execute `velocity.py` from the command line as a script

```
python velocity.py
```

- We can also include it into a different python file using the **`import`** command.
- An imported file is a Module.

Importing Modules

- A module is a file containing Python code that can be used by other modules or scripts
- A module is made available for use via the `import` statement
  - `import myModule`
- Once a module is imported, any feature defined in that module can be accessed using dot notation
  - `myModule.feature`

Modules vs. Scripts

- The following command can be used to set code that is only run when the file is a script.
- The tabbed in lines are only executed when the file is a script.

```
if __name__ == "__main__" :
    print("This file stores Physics Constants.")
    print("Gravity:", Gravity)
    print("EarthMass:", EarthMass)
    print("EarthRadius:", EarthRadius)
    print("m/s to miles/hour:", mps_to_milesph)
```

The Math Module

Python's math module supports more advanced math operations

Function	Description	Function	Description
ceil	Round up value	fabs	Absolute value
factorial	factorial (3! = 3 * 2 * 1)	floor	Round down value
fmod	Remainder of division	fsum	Floating-point sum
exp	Exponential function e^x	log	Natural logarithm
pow	Raise to power	sqrt	Square root
acos	Arc cosine	asin	Arc sine
atan	Arc tangent	atan2	Arc tangent with two parameters
cos	Cosine	sin	Sine
hypot	Length of vector from origin	degrees	Convert from radians to degrees
radians	Convert degrees to radians	tan	Tangent
cosh	Hyperbolic cosine	sinh	Hyperbolic sine
gamma	Gamma function	erf	Error function
pi (constant)	Mathematical constant 3.141592...	e (constant)	Mathematical constant 2.718281...

Math Module example

```
>>> import math
>>> n = math.sqrt(9.0)
>>> print(n)
3.0
>>> radius = 13.0
>>> area = math.pi * math.pow(radius, 2.0)
>>> print (area)
530.929158456675
>>>
```

Example

- We want to write a program that determines the escape velocity on different planets

- The formula for the escape velocity is: $\sqrt{\frac{2GM}{R}}$

Where:

- G is the Newton's gravitational constant
- M is the mass of the planet
- R is the radius of the planet

Example: using modules

```
import constants
import math

print("Escape Velocities")
vel = math.sqrt(2 * constants.Gravity *
    constants.MercuryMass / constants.MercuryRadius)
mph = vel * constants.mps_to_milesph
print("Mercury:", round(mph, 2), "miles/hour")
```