Topic 2
Expressions

CS 1MD3 • Introduction to Programming
Winter 2018

Dr. Douglas Stebila
Modified by Nicholas Moore
McMaster University

Python as a calculator

- We can do arithmetic directly in the Python interpreter or in Jupyter Notebook cells

- ```
  >>> 5 * 3
  15
  >>> 5 ** 3
  125
  >>> 2.5 / 3.7
  0.6756756756756757
  ```
  ```

Literal scalars

- Numbers like 3 and 2.5 in our source code are **objects**

- Specifically: literal scalar objects
  - **Object:** something we can operate on
  - **Scalar:** indivisible or atomic; not comprised of sub-components
  - **Literal:** represented directly in our source code, as opposed to being computed as the result of another operation

Literal Types

- `int`: represents positive & negative integers
  - Unlike other languages, no limits to size of an integer
  - Example int literals:
    `3 72 -56789`
- `float`: represents real numbers as floating point numbers
  - Floating point numbers cannot represent all real numbers and do not have perfect precision
  - Discussed in later weeks
  - Example float literals:
    `3.0 72.13 -56789.0123`

Operators and expressions

- **Operators** can be applied to objects to form **expressions** that yield other objects
  - Binary operators apply to two objects, e.g. `5 * 7`
  - Unary operators apply to one object, e.g. `-7`

- Example:
- ```
  >>> 5 * 7
  35
  ```
  ```

Querying Types

- We can use the `type` function to learn the type of an object
- ```
  >>> type(27)
  <class 'int'>
  >>> type(3.5)
  <class 'float'>
  ```
  ```

Binary operators on `int` and `float`

`a + b`	Addition
`a - b`	Subtraction
`a * b`	Multiplication
`a // b`	Integer division 7 // 4 is 1
`a / b`	Floating-point division 7/4 is 1.75
`a % b`	Modulus: "a mod b" Remainder from integer division 7 % 4 is 3
`a ** b`	Exponentiation

Quirks of int and float operators

- Rules of order of operations
  - BEDMAS generally applies
  - But safest to add your own parentheses to ensure you get the answer you want
- Can use different types of numbers together
- ```
  >>> type(5 * 3.0)
  <class 'float'>
  ```
- We can get unexpected answers due to imprecision of floating point representation
- >>> 0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+ 0.1+0.1
  ```

Common mathematical functions

- Many math functions are available
  - But not in the "core language"
  - Instead are available in the optional math **module**
- ```
  >>> import math
  >>> math.log2(256)
  8
  >>> math.cos(2 * math.pi)
  1.0
  ```

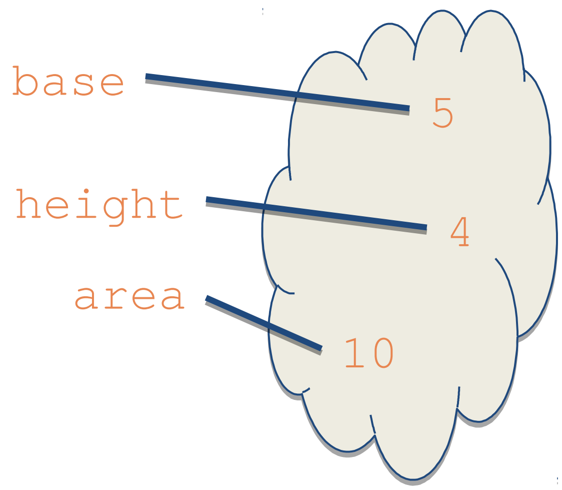
https://docs.python.org/3/

https://docs.python.org/3/library/math.html#module-math
  ```

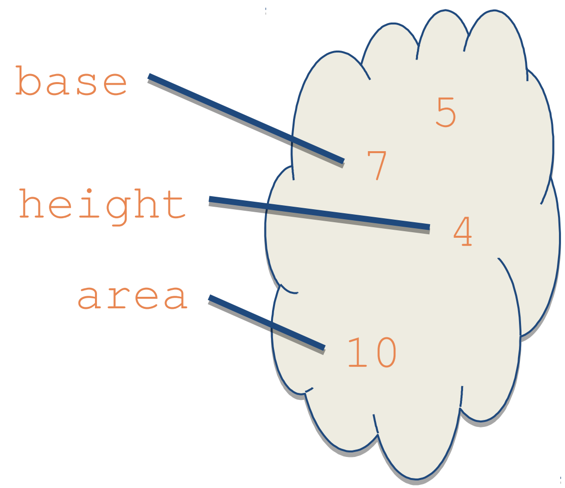
the variables is
0 or not are
depend on the
language

Variables

- **Variables** are names of objects




```
>>> base = 5
>>> height = 4
>>> area = base * height / 2
```




```
>>> base = 7
```

Variables

- Variables can contain uppercase and lowercase letter, digits, and _
- Variables must not start with a digit and must not be one of the reserved words (keywords):
  - def, if, while, return, and, import, global, …
  - You can see the full list of reserved keywords as follows:
    - `>>> import keyword`
    - `keyword.kwlist`

Assignment for variables

Assignment, written =, changes the state.

```
a = 4
```

- It is an instruction (command, statement) to change a value, not a check on whether two values are the same.
- We pronounce a = 4 as “a becomes 4”!

Later we will see `a == b` which means "check if the values are equal"

Multiple assignment

We can assignment values to multiple values at once

```
>>> x, y = 5, 3
```

```
>>> x, y = y, x
```

This will swap the values around

Note the difference from the sequence of commands

```
>>> x = y
>>> y = x
```

Question: What is the output of the following code?

```
>>> x, y = 7, 4
>>> x = y
>>> y = x
>>> x, y
```

A. 4, 7

B. 7, 7

C. 4, 4

Naming variables

- Try to use variable names that help you remember what role that object plays in your code

- `a = 3.14159`
- `b = 11.2`
- `c = a*(b**2)`

- `pi = 3.14159`
- `radius = 11.2`
- `area = pi*(radius**2)`

Comments

- You can add explanatory comments to your source code that won't get executed
- Use the # symbol to start a comment; anything after the # on the same line won't get executed
- Good source code includes lots of comments to help the reader understand the code

Comments

```
# Formula to calculate area of a rectangle
width = 3 # length of one side of the rectangle
height = 2 # length of the other side of the
           # rectangle
area = width * height
```

Booleans: True/False

- `bool` is another scalar object type in Python with two possible values: `True` and `False`

- ```
  >>> x = True
  >>> type(x)
  <class 'bool'>
  ```
  ```

Operators related to Boolean values

- **Comparators** will compare two ints/floats and yield a Boolean

a == b	Equality: Do a and b have the same value?
a <= b	Less than or equal to
a < b	Less than but not equal to
a >= b	Greater than or equal to
a > b	Greater than
a != b	Not equal to

Equality (==) versus assignment (=)

- Be careful about = versus ==

- = means assignment: it is a statement, an instruction, it updates the state
- `>>> a = 4`
- The variable a now contains 4

- == means equality: it is a test, a check, it evaluates to either True or False
- `>>> a == 4`
- The result is either True or False
- The contents of the variable a are unchanged

Functions

- A way of grouping together a sequence of operations under a common name so we can refer to it multiple times later
- Similar to mathematical functions: cos, exp, log, etc.

Syntax of functions

def used to define a function

Same rules for naming functions as for variables

0 or more variable names, which will be assigned to whatever values the function caller provides

```
def name_of_function(list_of_inputs):
    body_of_function
    return value_to_return
```

The output to give back to the function caller

Body and return statement have to be indented with one tab character

Example function

```
def pythagoras(x, y):
  z = (x ** 2) + (y ** 2)
  return z ** 0.5

>>> pythagoras(3, 4)
5
>>> a = 4
>>> b = 3
>>> c = pythagoras(a, b)
>>> c == 5
True
```

Multiplication versus calling functions

- What happens when we type the following code:
- >>> a = 4
- >>> b = 5
- >>> 3(a+b)

Tuples

- Tuples are a list of objects
```
>>> (5, 7, 9)
```
- The objects in the list don't have to have the same type:
```
>>> (4, 7.0, "potato", True)
```
- We can put any number of items we want into a tuple
- They stay in order
- We can assign a tuple to a variable:
```
>>> T = (5, 7, 9)
```

Operations on tuples

T[i]	Retrieves the i+1'th element of the tuple
T + U	Concatenation Yields a new tuple containing all the items in T followed by all the items in U
len(T)	Returns the number of elements in the tuple
x in T	True if x is an element in the tuple T False otherwise
x not in T	True if x is not an element in the tuple T False otherwise

Interesting observations about tuples

- Tuples are **immutable:** we can't change one entry of a tuple
- We can use
  ```
  >>> x = T[3]
  ```
  to read out the 4^{th} entry of tuple T
- Important: tuples are "0-indexed": we start counting at T[0], T[1], ..., T[n-1]
- But we can't use
  ```
  >>> T[3] = 7
  ```
  to set the 4^{th} entry of tuple T

Strings

- Strings are a sequence of characters
  - Like a tuple

```
>>> s = "Hello, world!"
>>> s
'Hello, world!'
>>> type(s)
<class 'str'>
```

String literals

- String literals: strings that you type directly into your source code
- Have to be wrapped in either single quotation marks '...' or double quotation marks "..."
- ```
  >>> s = Hello, world!
  >>> s = "Hello, world!"
  >>> s = 'Hello, world!'
  ```
  ```

String literals

- What if you want to include a single or double quotation mark in your string?

- One solution: use the alternate type of quotation mark to contain it:
```
>>> s = 'Hello, "Alice"!'
```

- Best solution: use backslash \ to escape:
```
>>> s = "Hello, \"Alice\"!"
```

Characters

- What type of letters can we use in our strings?
- Historically, most languages only allowed ASCII characters
  - ASCII: dates from the 1960s, used originally on teletype machines
  - Each character was represented by 7 bits
    - Total possible # of characters: $2^7 = 128$

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char	Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*	48	30	110000	60	0	96	60	1100000	140	`
1	1	1	1	*[START OF HEADING]*	49	31	110001	61	1	97	61	1100001	141	a
2	2	10	2	*[START OF TEXT]*	50	32	110010	62	2	98	62	1100010	142	b
3	3	11	3	*[END OF TEXT]*	51	33	110011	63	3	99	63	1100011	143	c
4	4	100	4	*[END OF TRANSMISSION]*	52	34	110100	64	4	100	64	1100100	144	d
5	5	101	5	*[ENQUIRY]*	53	35	110101	65	5	101	65	1100101	145	e
6	6	110	6	*[ACKNOWLEDGE]*	54	36	110110	66	6	102	66	1100110	146	f
7	7	111	7	*[BELL]*	55	37	110111	67	7	103	67	1100111	147	g
8	8	1000	10	*[BACKSPACE]*	56	38	111000	70	8	104	68	1101000	150	h
9	9	1001	11	*[HORIZONTAL TAB]*	57	39	111001	71	9	105	69	1101001	151	i
10	A	1010	12	*[LINE FEED]*	58	3A	111010	72	:	106	6A	1101010	152	j
11	B	1011	13	*[VERTICAL TAB]*	59	3B	111011	73	;	107	6B	1101011	153	k
12	C	1100	14	*[FORM FEED]*	60	3C	111100	74	<	108	6C	1101100	154	l
13	D	1101	15	*[CARRIAGE RETURN]*	61	3D	111101	75	=	109	6D	1101101	155	m
14	E	1110	16	*[SHIFT OUT]*	62	3E	111110	76	>	110	6E	1101110	156	n
15	F	1111	17	*[SHIFT IN]*	63	3F	111111	77	?	111	6F	1101111	157	o
16	10	10000	20	*[DATA LINK ESCAPE]*	64	40	1000000	100	@	112	70	1110000	160	p
17	11	10001	21	*[DEVICE CONTROL 1]*	65	41	1000001	101	A	113	71	1110001	161	q
18	12	10010	22	*[DEVICE CONTROL 2]*	66	42	1000010	102	B	114	72	1110010	162	r
19	13	10011	23	*[DEVICE CONTROL 3]*	67	43	1000011	103	C	115	73	1110011	163	s
20	14	10100	24	*[DEVICE CONTROL 4]*	68	44	1000100	104	D	116	74	1110100	164	t
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*	69	45	1000101	105	E	117	75	1110101	165	u
22	16	10110	26	*[SYNCHRONOUS IDLE]*	70	46	1000110	106	F	118	76	1110110	166	v
23	17	10111	27	*[ENG OF TRANS. BLOCK]*	71	47	1000111	107	G	119	77	1110111	167	w
24	18	11000	30	*[CANCEL]*	72	48	1001000	110	H	120	78	1111000	170	x
25	19	11001	31	*[END OF MEDIUM]*	73	49	1001001	111	I	121	79	1111001	171	y
26	1A	11010	32	*[SUBSTITUTE]*	74	4A	1001010	112	J	122	7A	1111010	172	z
27	1B	11011	33	*[ESCAPE]*	75	4B	1001011	113	K	123	7B	1111011	173	{
28	1C	11100	34	*[FILE SEPARATOR]*	76	4C	1001100	114	L	124	7C	1111100	174	|
29	1D	11101	35	*[GROUP SEPARATOR]*	77	4D	1001101	115	M	125	7D	1111101	175	}
30	1E	11110	36	*[RECORD SEPARATOR]*	78	4E	1001110	116	N	126	7E	1111110	176	~
31	1F	11111	37	*[UNIT SEPARATOR]*	79	4F	1001111	117	O	127	7F	1111111	177	*[DEL]*
32	20	100000	40	*[SPACE]*	80	50	1010000	120	P					
33	21	100001	41	!	81	51	1010001	121	Q					
34	22	100010	42	"	82	52	1010010	122	R					
35	23	100011	43	#	83	53	1010011	123	S					
36	24	100100	44	$	84	54	1010100	124	T					
37	25	100101	45	%	85	55	1010101	125	U					
38	26	100110	46	&	86	56	1010110	126	V					
39	27	100111	47	'	87	57	1010111	127	W					
40	28	101000	50	(	88	58	1011000	130	X					
41	29	101001	51	)	89	59	1011001	131	Y					
42	2A	101010	52	*	90	5A	1011010	132	Z					
43	2B	101011	53	+	91	5B	1011011	133	[					
44	2C	101100	54	,	92	5C	1011100	134	\					
45	2D	101101	55	-	93	5D	1011101	135	]					
46	2E	101110	56	.	94	5E	1011110	136	^					
47	2F	101111	57	/	95	5F	1011111	137	_					

https://commons.wikimedia.org/wiki/File:ASCII-Table.svg

Unicode

- ASCII didn't have enough spots to represent accented characters and other languages
  - 8-bit ASCII supported 256 bit characters, but still not enough
- Unicode: International standard that specified characters in hundreds of languages
  - Currently 136,755 characters in the Unicode spec
  - Not all fonts have symbols for all characters
- UTF-8: A way of representing Unicode characters in 8-bit bytes
  - The first 128 characters of UTF-8 Unicode match the first 128 characters of ASCII
  - Higher Unicode characters take up more bytes in UTF-8 representation

Unicode

- All Python3 files are UTF-8 by default
- This means you can use arbitrary Unicode characters in string literals directly
-
  ```
  >>> s = "Je m'appelle Léna."
  >>> s = "My last name is 张".
  >>> s = "Have fun! 😜"
  ```

- You can use \n inside a string literal to include a new line

  ```
  >>> s = "Dear John,\nHave a good day."
  ```

Operators on strings

Operator	Notation
indexing	`S[i]`
suffix, starting at `i`	`S[i:]`
prefix, not including `j`	`S[:j]`
slice, starting at `i`, not including `j`	`S[i:j]`
length	`len(S)`
occurrences of a substring	`S.count(E)`
first index of a subtring	`S.index(E)`

```
>>> s = 'CDEFGABC'
>>> s[1], s[-2], s[1:], s[:2], s[3:5], len(s)
>>> s.count('AB'), s.index('C')
```

In general:

```
S[i] = S[i%len(S)] = S[i+len(S)] if -len(S) <= i < 0
S[i:] = S[i:len(S)]
S[:j] = S[0:j]
```

Strings are immutable

- Immutable: can't change individual characters without creating a new string

- ```
  >>> s = "Hello, world!"
  >>> s[3] = "p"
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  TypeError: 'str' object does not support
  item assignment
  ```
  ```

More operators on strings

Operator	Notation
concatenation	`S+T`
substring	`E in S, E not in S`
repetition i times	`S*i, i*S`
suffix test	`S.endswith(T)`

```
('Nah'+' nah'*2+(' nah'*3+',')*2+' hey Jude\n')*16

def hasUndefinedBase(s):
   return 'n' in s

>>> hasUndefinedBase('ggacntgtc')
```

Printing values

- `print(`*`expression`*`)` evaluates the given expression and then outputs it to the screen/console

Getting input

- y = input(x) will display the string x, let the user type in a value, and then save that value in the variable y

- ```
  >>> y = input("Enter your name: ")
  >>> print("Hello, " + y)
  ```
  ```

String formatting

- You can use the .format to format numbers and other values as strings
- The **format string** is constructed using normal characters, as well as special **format specifiers**

```
>>> 'Hello, {:s}'.format("Bob")
'Hello, Bob'
>>> 'The year is {:d}'.format(2018)
'The year is 2018'
>>> 'pi to three decimal places is {:.3f}'.format(math.pi)
 'pi to three decimal places is 3.142'
>>> 'Happy {:d}, {:s}'.format(2018, "Bob")
```

https://docs.python.org/3.3/library/string.html#format-string-syntax

More string examples

```
>>> def frequency(s, t):
        return s.count(t)/len(s)*100
>>> dna = 'tatgaatggactgtccccaaagaagtagga'
>>> frequency(dna, 't')

>>> def plural(w):
        return w+'es' if w.endswith('s') 
else \
               w+'s'
>>> plural('duck')
>>> plural('walrus')
```

Conditional operator

Operator	Notation
conditional	`E if B else F`

The **conditional** evaluates the left operand if the condition B, a Boolean expression is True, otherwise the right operand; E, F are of arbitrary types:

```
E if True else F = E
E if False else F = F
```

As a consequence:

```
E if not B else F = F if B else E
E if B else E = E
```

Defining the maximum of a pair by a conditional expression:

```
def maximum(a, b):
return a if a > b else b
```

Boolean operators

and

- "B and C" is True if and only if both B and C evaluate to True

```
>>> B and C
```

or

- "B or C" is True if and only if either B is True, or C is True, or both are True

```
>>> B or C
```

xor

- "Exclusive or"
- "B xor C" is True if and only if one of them is True and the other is False

```
>>> B ^ C
```

Boolean operators

not

- "not B" gives the opposite of B
- not True -> False
- not False -> True

- `>>> not B`

Formulas involving Booleans

- not, and, and or interact in different ways
- not(B and C) = (not B) or (not C)
  - B = I ate ice cream
  - C = I ate ketchup
  - B and C = I ate ice cream and ketchup
  - not(B and C) = I did not eat ice cream and ketchup
  - (not B) = I did not eat ice cream
  - (not C) = I did not eat ketchup
  - (not B) or (not C) = either I did not eat ice cream, or I did not eat ketchup
- This is one of De Morgan's laws

Question: What is Z?

```
>>> X = True
>>> Y = False
>>> Z = not(not(X and not(Y)) or
Y)
```

A. True

B. False