Week 2

Intro to Python

Chapters 2 & 3

Topics for today:

- Input function
- Eval function
- Simultaneous assignments
- Identifiers
- Numeric operators
- Data type conversions and rounding
- Built-in functions
- ASCII Code
- Strings
- Formatting numbers and strings

input() and eval()

We can ask the user to input a value.

A simple example is to ask the user to input their name and then display it.

```
# Name of user
name = input("Enter your name: ")

# Displays name
print("Your name is", name)
```

Suppose we have three circles and we want to find their area. We have found the radii to be 3 in, 7 in, and 9 in and we write a program that will quickly calculate these areas.

```
# First we define pi
pi = 3.1416

# Then we ask the user to input the radius
radius = eval(input("Enter the radius: "))

# Calculate area
area = (radius**2)*pi

# Display the answer
print("The area of the circle is", area, "in^2")
```

Simultaneous assignments

Since we already knew we were going to have three circles we can ask for the three radii at once.

```
# Pi
pi = 3.1416

# Input radii
r1, r2, r3 = eval(input("Enter the radii separated by commas: "))

# Calculate areas
a1, a2, a3 = (r1**2)*pi, (r2**2)*pi, (r3**2)*pi

# Display the answers
print("First area is", a1, "in^2")
print("Second area is", a2, "in^2")
print("Third area is", a3, "in^2")
```

Identifiers

An identifier is a sequence of characters that consists of letters, digits, and underscores (_).

It cannot begin with a digit.

It also cannot be a keyword.

Functions and variables are examples of identifiers:

my_variable

print

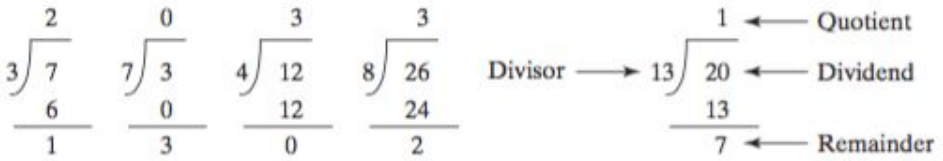
List of keywords

and, else, as, except, assert, False, break, finally, class, for, continue, from, def, global, or, del, if, pass, elif, import, raise, in, return, is, True, lambda, try, None, while, nonlocal, with not, yield.

Numeric operators

Name	*Meaning*	*Example*	*Result*
`+`	Addition	`34 + 1`	`35`
`-`	Subtraction	`34.0 - 0.1`	`33.9`
`*`	Multiplication	`300 * 30`	`9000`
`/`	Float Division	`1 / 2`	`0.5`
`//`	Integer Division	`1 // 2`	`0`
`**`	Exponentiation	`4 ** 0.5`	`2.0`
`%`	Remainder	`20 % 3`	`2`

Perhaps the most confusing operator is the modulo (%) or “mod” operator. It returns the remainder after the operation.



What would be the answer for 42 % 5?
42 % 4?

Conversions and rounding

Converting from floats to integers.

Float: 5.6

```
int(5.6)
```

Yields 5, *int()* truncates the number.

Rounding floats.

Float: 5.6

```
round(5.6)
```

Yields 6. If we use the function to round 5.3 we would get *5*.

You can round to a specific number of digits.

```
x = 43/37
round(x, 2)
>>> 1.16
round(x, 5)
>>> 1.16216
```

Built-in functions

Simple 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

These can be used by importing the math module, just type `import math`.

The math module has helpful functions like square-roots and logarithmic functions, as well as numbers like π or e (Euler's number).

```
import math

# natural log
math.log(10)

# square root
math.sqrt(9)

# log(x, base)
math.log(100, 10)

# pi, 3.141592653589793
math.pi

# e, 2.718281828459045
math.e
```

ASCII

American **S**tandard **C**ode for **I**nformation **I**nterchange

Computers can only understand numbers

ASCII code is the numerical representation of all uppercase and lowercase letters, digits, punctuation marks, and control characters

ord(‘a’) = 97

➔ Returns ASCII code value

chr(97) = ‘a’

➔ Returns character, given ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Strings

A sequence of characters and can include text and numbers. String values are enclosed in matching single quotes (') or double quotes (").

What if we wanted to include double quotes in our print statement? We can use escape sequences.

Python Escape Sequences

Character Escape Sequence	*Name*
\b	Backspace
\t	Tab
\n	Linefeed
\f	Formfeed
\r	Carriage Return
\\	Backslash
\'	Single Quote
\"	Double Quote

```
# The following will give you an error
print("The instructor said "Python is fun!"")

# Correct way to print this
print("The instructor said \"Python is fun!\"")
```

We can convert a number to string using the *str()* function:

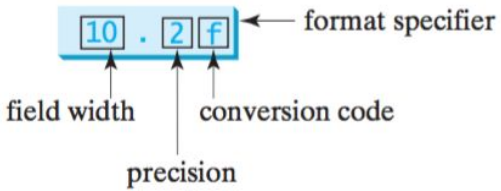
```
# float to string
str(3.4)

# int to string
str(32)
```

To concatenate strings we can use the String Concatenation Operator or the + Operator

Formating

We can format numbers and strings to be in a specific width and to a certain precision.



The specifiers for numbers are:

Floats - *f*

Integers - *d* (decimal), *x* (hexadecimal), and *b* (binary)

Strings - s

Some examples:

```
# the string will be length 8
# the precision is to 3 digits
# past the decimal point
format(23.1245, "8.3f")
>>> '  23.125'

format(23, "10.2f")
>>> '     23.00'

format(3452, "5d")
>>> ' 3452'

format("Intro to Python", "20s")
>>> 'Intro to Python     '
```

Formatting (cont.)

Other specifiers:

Scientific notation: *e*

Percentage: %

TABLE 3.4 Frequently Used Specifiers

Specifier	*Format*
`"10.2f"`	Format the float item with width 10 and precision 2.
`"10.2e"`	Format the float item in scientific notation with width 10 and precision 2.
`"5d"`	Format the integer item in decimal with width 5.
`"5x"`	Format the integer item in hexadecimal with width 5.
`"5o"`	Format the integer item in octal with width 5.
`"5b"`	Format the integer item in binary with width 5.
`"10.2%"`	Format the number in decimal.
`"50s"`	Format the string item with width 50.
`"<10.2f"`	Left-justify the formatted item.
`">10.2f"`	Right-justify the formatted item.

More examples:

```
format(8267392, "<10.2e")
>>>'8.27e+06  '

format(8267392, "10.2e")
>>>'  8.27e+06'

format(.34, "5.1%")
>>>'34.0%'

format(43, "<10b")
>>>'101011    '
```

More details in section 3.6.1 of the textbook