1. Basics

Topics:
- Interactive Mode
- Variables
- Expressions
- The Assignment Statement
- Types

The Python Interactive Shell

Python can be used in a way that reminds you of a calculator. In the ``command shell'' of your system simply type

```
python
```

and you will be met with a prompt…

```
>>>
```

Let's Compute the Area of a Circle Using Python

```
>>> r = 10
>>> A = 3.14*r*r
>>> print A
314.0
```

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r*r
>>> print A
314.0
```

Notation is different.
In Python, you can't say A = 3.14xrxr

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r**2
>>> print A
314.0
```

Notation is different.
You can indicate exponentiation with **

Programming vs Math

```
>>> r = 10
>>> A = 3.14*r**2
>>> print A
314.0
```

`r` and `A` are variables. In algebra, we have the notion of a variable too. But there are some big differences.

Variables

```
>>> r = 10
>>> A = 3.14*r**2
```

r -> 10   A -> 314.0

A variable is a named memory location. Think of it as a box.
It contains a value. Think of it as the contents of the box.

" The value of r is 10. The value of A is 314.0."

The Assignment Statement

```
>>> r = 10
```

r -> 10

The "=" symbol indicates assignment
The assignment statement r = 10 creates the variable r and assigns the value of 10 to it.

Formal: " r is assigned the value of 10"   Informal: "r gets 10"

The Assignment Statement

```
>>> r = 10
>>> A = 3.14*r**2
```

r -> 10   A -> 314.0

A variable can be used in an expression like
`3.14*r**2`.
The expression is evaluated and then stored.

Assignment Statement: WHERE TO PUT IT = RECIPE FOR A VALUE

Order is Important

```
>>> A = 3.14*r**2
>>> r = 10
NameError: name 'r' is not defined
```

Math is less fussy:
A = 3.14*r**2 where r = 10

Assignment vs. "Is Equal to"

```
>>> r = 10
>>> 3.14*r**2 = A
SyntaxError: can't assign to an
operator
```

In Math "=" is used to say what is on the left equals what is on the right.
In Python, "=" prescribes an action, "evaluate the expression on the right and assign its value to the variable named on the left."

The Assignment Statement

```
>>> r = 10
>>> A = 3.14*r**2
>>> S = A/2
```

r -> 10
A -> 314.0
S -> 157.0

Here we are assigning to S the area of a semicircle that has radius 10.

Assignment Statement: WHERE TO PUT IT = RECIPE FOR A VALUE

The Assignment Statement

```
>>> r = 10
>>> A = 3.14*r**2
>>> A = A/2
```

```
 r -> 10
A -> 157.0
```

Here we are assigning to A the area of a semicircle that has radius 10.

Nothing new in the third assignment. The "recipe" is A/2. The target of the assignment is A.

"A has been overwritten by A/2"

Tracking Updates

```
>>> y = 100
```

Before:

Tracking Updates

```
>>> y = 100
```

After:

```
y -> 100
```

Tracking Updates

```
>>> y = 100
>>> t = 10
```

Before:

```
y -> 100
```

Tracking Updates

```
>>> y = 100
>>> t = 10
```

After:

```
y -> 100

t -> 10
```

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
```

Before:

```
y -> 100

t -> 10
```

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
```

After:

y -> 110

t -> 10

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
```

Before:

y -> 110

t -> 10

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
```

After:

y -> 110

t -> 20

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10;
>>> y = y+t
```

Before:

y -> 110

t -> 20

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10;
>>> y = y+t
```

After:

y -> 130

t -> 20

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
>>> y = y+t
>>> t = t+10
```

Before:

y -> 130

t -> 20

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
>>> y = y+t
>>> t = t+10
```

After:

y -> 130

t -> 30

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
>>> y = y+t
>>> t = t+10
>>> y = y+t
```

Before:

y -> 130

t -> 30

Tracking Updates

```
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
>>> y = y+t
>>> t = t+10
>>> y = y+t
```

After:

y -> 160

t -> 30

Assignment vs Equations

In algebra,

t = t +10

doesn't make sense unless you believe that
0 =t-t = 10

In Python,

```
t = t + 10
```

means add 10 to the value of t and store the result in t.

The Key 2-Step Action Behind Every Assignment Statement

< *variable name* > = < *expression* >

1. Evaluate the expression on the right hand side.

2. Store the result in the variable named on the left hand side.

Naming Variables

```
>>> radius = 10
>>> Area = 3.14*radius**2
```

radius -> 10   Area -> 314.0

Rule 1. Name must be comprised of digits, upper case letters, lower case letters, and the underscore character "_"
Rule 2. Must begin with a letter or underscore

A good name for a variable is short but suggestive of its role: `Circle_Area`

Precedence

Q. In an arithmetic expression, what is the order of evaluation?

A. Exponentiation & negation comes before multiplication & division which in turn come before addition & subtraction

This:	Is the same as:
`A + B*C`	`A + (B*C)`
`-A**2/4`	`-(A**2)/4`
`A*B/C*D`	`((A*B)/C)*D`

It is a good habit to use parentheses if there is the slightest ambiguity.

Revisit Circle Area

```
>>> r = 10
>>> A = (22/7)*r**2
>>> print A
300.0
```

It seems that Python evaluates (22/7) as 3 instead of 3.142… WHY?

A different kind of arithmetic. We have a related experience here.
11+3 = 2 in "clock arithmetic"

Integers and Decimals

In math we distinguish between integer numbers and decimal numbers.

Integer Numbers:
100, 0, -89, 1234567

Decimal Numbers:
-2.1, 100.01, 100.0, 12.345

Integers and Decimals

Different kinds of division

Integer Division:
30/8 is 3 with a remainder of 6

Decimal Division:
30/8 is 3.75

`int` vs `float`

In Python, a number has a type.

The `int` type represents numbers as integers.

The `float` type represents numbers as decimals.

Important to understand the differences and the interactions

`int` Arithmetic

```
>>> x = 30
>>> y = 8
>>> q = x/y
>>> print q
3
>>> r = x%y
>>> print r
6
```

To get the remainder, use %. Python "knows" that the values stored in x and y have type int because there are no decimal points in those assignments.

float Arithmetic

```
>>> x = 30.
>>> y = 8.
>>> q = x/y
>>> print q
3.75
```

Python "knows" that the values stored in x and y have type float because there are decimal points in those assignments.

Mixing float and int

```
>>> x = 30.
>>> y = 8
>>> q = x/y
>>> print q
3.75
```

In Python if one operand has type float and the other has type int, then the type int value is converted to float and the evaluation proceeds.

Explicit Type Conversion

```
>>> x = 30.0
>>> y = 8.0
>>> q = int(x)/int(y)
>>> print q
3
```

`int( -`*expression*`- )` converts the value of the expression to `int` value

Explicit Type Conversion

```
>>> x = 30
>>> y = 8
>>> q = float(x)/float(y)
>>> print q
3.75
```

`float( -`*expression*`- )` converts the value of the expression to a `float`

An Important Distinction

Integer arithmetic is exact.
Float arithmetic is (usually) not exact

```
>>> x = 1.0/3.0
>>> print x
.333333333333
```

Strings

So far we have discussed computation with numbers.

Now we discuss computation with text.

We use strings to represent text.

You are a "string processor" when you realize 7/4 means July 4 and not 1.75!

Strings

Strings are quoted characters. Here are three examples:

```
>>> s1 = 'abc'
>>> s2 = 'ABC'
>>> s3 = ' A B C '
```

s1,s2,and s3 are variables with string value.

Strings

Strings are quoted characters. Here are three examples:

```
>>> s1 = 'abc'
>>> s2 = 'ABC'
>>> s3 = ' A B C '
```

The values in s1,s2,and s3 are all different. Upper and lower case matters. Blanks matter

Strings

Nothing special about letters…

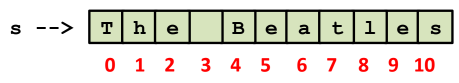
```
>>> Digits = '1234567890'
>>> Punctuation = '!:;.?'
>>> Special = @#$%^&*()_-+='
```

Basically any keystroke but there are some exceptions and special rules. More later.

Here is one : 'Sophie"'"s Choice' i.e., Sophie's Choice

Strings are Indexed

```
>>> s = 'The Beatles'
```

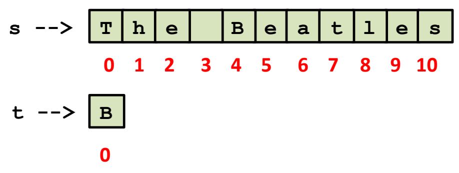


The characters in a string can be referenced through their indices. Called "subscripting".

Subcripting from zero creates a disconnect: 'T' is not the first character.

Strings are Indexed

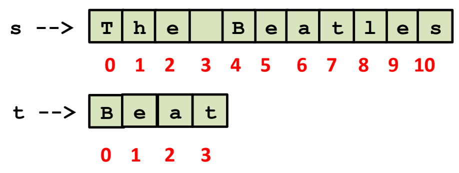
```
>>> s ='The Beatles'
>>> t = s[4]
```



The square bracket notation is used. Note, a single character is a string.

Strings are Indexed

```
>>> s ='The Beatles'
>>> t = s[4:8]
```



We say that "t is a slice of s".

Strings are Indexed

```
>>> s ='The Beatles'
>>> t = s[4:]
```

s -->

T	h	e		B	e	a	t	l	e	s
0	1	2	3	4	5	6	7	8	9	10

t -->

B	e	a	t	l	e	s
0	1	2	3	4	5	6

Same as `s[4:11]`. Handy notation when you want an "ending slice."

Strings are Indexed

```
>>> s ='The Beatles'
>>> t = s[:4]
```

s -->

T	h	e		B	e	a	t	l	e	s
0	1	2	3	4	5	6	7	8	9	10

t -->

T	h	e	
0	1	2	3

Same as `s[0:4]`. Handy notation when you want a "beginning slice".

Strings are Indexed

```
>>> s ='The Beatles'
>>> t = s[11]
IndexError: string index out of
range
```

s -->

T	h	e		B	e	a	t	l	e	s
0	1	2	3	4	5	6	7	8	9	10

The is no s[11]. An illegal to access.

Subscripting errors are EXTREMELY common.

Strings Can Be Combined

```
>>> s1 = 'The'
>>> s2 = 'Beatles'
>>> s = s1+s2
```

s -->

T	h	e	B	e	a	t	l	e	s

This is called concatenation.

Concatenation is the string analog of addition except

Concatenation

```
>>> s1 = 'The'
>>> s2 = 'Beatles'
>>> s = s1 + ' ' + s2
```

s -->

T	h	e		B	e	a	t	l	e	s

We "added" in a blank.

No limit to the number of input strings: `s = s2+s2+s2+s2+s2`

Types

Strings are a type: `str`

So at this point we introduced 3 types:

`int`	for integers, e.g., `-12`
`float`	for decimals, e.g., `9.12`, `-12.0`
`str`	for strings, e.g., `'abc'`, `'12.0'`

Python has other built-in types. And we will learn to make up our own types.

A Type is a Set of Values and Operations on Them

Values…

`int`	`123, -123, 0`
`float`	`1.0, -.00123, -12.3e-5`
`str`	`'abcde', '123.0'`

⇧ ⇧ ⇧ ⇧ ⇧

These are called "literals"

The "e" notation (a power-of-10 notation) is handy for very large or very small

A Type is a Set of Values and Operations on Them

Operations…

`int`	`+ - * / unary- ** %`
`float`	`+ - * / unary- **`
`str`	`+`

⇧

concatenation

Type Conversion

```
>>> s = '123.45'
>>> x = 2*float(s)
>>> print x
246.90
```

A string that encodes a decimal value can be represented as a **`float`**.

Type Conversion

```
>>> s = '-123'
>>> x = 2*int(s)
>>> print x
-246
```

A string that encodes an integer value can be represented as an **`int`**.

Type Conversion

```
>>> x = -123.45
>>> s = str(x)
>>> print s
'-123.45'
```

Shows how to get a string encoding of a float value.

Automatic Type Conversion

```
>>> x = 1/2.0
>>> y = 2*x
```

An operation between a **`float`** and an **`int`** results in a **`float`**. So **`x`** is a **`float`**.

Thus, **`y`** is also a **`float`** even though its value happens to be an integer.

Python is a Dynamically Typed Language

A variable can hold different types of values at different times.

```
>>> x = 'abcde'
>>> x = 1.0
>>> x = 32
```

In other languages the type of a variable is fixed

Summary

1. Variables house values that can be accessed.

2. Assignment statements assign values to variables.

3. Numerical data can be represented using the `int` and `float` types.

4. Text data can be represented using the `str` type.

Terminology

assignment statement

A statement of the following form:

<variable> = *<value>*

If the variable does not yet exist, the statement creates the variable and stores the given value inside it. If the variable does exist, then it replaces the old value with the one provided.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

float

A primitive type whose values are scientific numbers, e.g. 3.46E–4.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

int

A primitive type whose values are integers.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

interactive shell

A program that allows the user to type in Python expressions and statements one at a time and evalautes them.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

literal

A Python denotation of a value. Literals are used to initialize any of the primitive types, or initialize Strings. Examples of literals:

int	354
float	3.56
boolean	True
String	'Hello World!'

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

precedence

Standard mathematics gives precedence to multiplication * over addition +, so that the expression 2 + 3 * 5 is evaluated as if it were parenthesized like this: 2 + (3 * 5). The precedences used in Python for all operators are given on this page.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

type

A set of values together with operations on them. The type of a value can be determined via the type() function.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php

Terminology

variable

A name with an associated value. Sometimes viewed as a named box with the value in the box. In Python, there are four kinds of variable: parameter, local variable, attribute, and a global variable.

Reference: http://www.cs.cornell.edu/Courses/cs1110/2015sp/materials/definitions.php