1. The Assignment Statement and Types

Topics:
- Python’s Interactive Mode
- Variables
- Expressions
- Assignment
- Strings, Ints, and Floats

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 $A = 3.14r^2$

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

Programming vs Math

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

Notation is different.
In Python you indicate exponentiation with **

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

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

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

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

```
r -> 10  A -> 314.0
```

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

The Assignment Statement

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

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

```
r -> 10
```

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

The Assignment Statement

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

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

```
r -> 10  A -> 314.0
```

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

Order is Important

Math is less fussy: 

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

NameError: name 'r' is not defined

Assignment vs. “Is Equal to”

In Math “=” is used to say what is on the left equals what is on the right.

In Python, “=” prescribing an action, “evaluate the expression on the right and assign its value to the variable named on the left.”

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

The Assignment Statement

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

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

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

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

Here we are assigning to \( A \) the area of a semicircle that has radius 10.
No new rules in the third assignment. The "recipe" is \( A/2 \). The target of the assignment is \( A \).

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

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

"A has been overwritten by A/2."

Tracking Updates

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

Before:

```
y -> 100
```

After:

```
After:

>>> t = 10
```

```
y -> 100
t -> 10
```
Tracking Updates

```python
>>> y = 100
>>> t = 10
>>> y = y+t
After:
y -> 110
t -> 10
```

Before:

```python
>>> y = 100
>>> t = 10
>>> y = y+t
>>> t = t+10
```
Tracking Updates

```python
>>> y = 100
>>> t = 10
>>> y = y + t
>>> t = t + 10
>>> y = y + t
>>> t = t + 10
After:
y -> 130
t -> 30
```

Tracking Updates

```python
>>> y = 100
>>> t = 10
>>> y = y + t
>>> t = t + 10
>>> y = y + t
>>> t = t + 10
Before:
y -> 130
t -> 30
```

Assignment vs Equations

In algebra,

\[ t = t + 10 \]

doesn't make sense unless you believe

\[ 0 = t - t = 10 \]

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

\[
< \text{variable name} > = < \text{expression} >
\]

1. Evaluate the expression on the right hand side.
2. Store the result in the variable named on the left hand side.

Naming Variables

```python
>>> 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.

<table>
<thead>
<tr>
<th>This:</th>
<th>Is the same as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A + B*C</td>
<td>A + (B*C)</td>
</tr>
<tr>
<td>-A**2/4</td>
<td>-(A**2)/4</td>
</tr>
<tr>
<td>A<em>B/C</em>D</td>
<td>((A*B)/C)*D</td>
</tr>
</tbody>
</table>

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

**Revisit Circle Area**

```python
>>> 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

**int vs float**

In Python, a number has a **type**.

The **int** type represents numbers as integers.

The **float** type represents numbers as decimals.

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 these assignments.

```python
>>> x = 30
>>> y = 8
>>> q = x/y
>>> print q
3
>>> r = x%y
>>> print r
6
```
**float Arithmetic**

```python
>>> 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**

```python
>>> 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**

```python
>>> 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*

```python
>>> 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.

```python
>>> 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

Strings are quoted characters. Here are three examples:

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

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

Strings

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

Strings

Nothing special about letters…

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

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

Strings are Indexed

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

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

```python
s[4] --> B
```

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

Strings are Indexed

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

```python
>>> t = s[4]
```

We say that "$t$ is a slice of $s$".
String Slicing

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

`s --> The Beatles`

```
0 1 2 3 4 5 6 7 8 9 10
```

`t --> Beatles`

```
0 1 2 3 4 5 6
```

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

---

String Slicing

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

`s --> The Beatles`

```
0 1 2 3 4 5 6 7 8 9 10
```

`t --> The`

```
0 1 2 3
```

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

---

String Slicing

```python
>>> s = 'The Beatles'
>>> t = s[11]
```

IndexError: string index out of range

`s --> The Beatles`

```
0 1 2 3 4 5 6 7 8 9 10
```

`t --> `

```
0 1 2
```

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

---

String Slicing

```python
>>> s = 'The Beatles'
>>> t = s[8:20]
```

`s --> The Beatles`

```
0 1 2 3 4 5 6 7 8 9 10
```

`t --> les`

```
0 1 2
```

It is "OK" to shoot beyond the end of the source string.

---

Strings Can Be Combined

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

`s --> TheBeatles`

This is called concatenation.

---

Concatenation

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

`s --> The Beatles`

We "added" in a blank.
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`, `-0.0123`, `-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 floats.
The literals `-0.00123` and `-12.3e-5` are the same number.

Operations...

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

Type Conversion

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

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

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

Shows how to get a string encoding of a float value.
**Automatic Type Conversion**

```python
>>> 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.

```python
>>> 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.