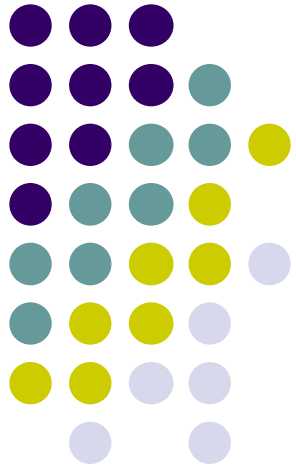
ISYS90088 Introduction to Application Development



Week 2 – Software Development Life Cycle

Variable, data types - integers and floating point numbers, construct arithmetic expressions

Initialize and use variables

Semester 2 , 2017

Dr Thomas Christy

Consolidate Last week

- Introduced the subject
- Assessments
- Topics of class
- IDLE - GROK
- Python
  - Algorithms
  - Syntax
    - Print
    - String “ ”
    - Int
    - Input
    - Variables

Objectives

After completing this Lecture, you will be able to:

- Describe the basic phases of software development: analysis, design, coding, and testing
- Variables, data types - use integers and floating point numbers in arithmetic operations
- Construct arithmetic expressions
- Initialize and use variables with appropriate names
- Use strings for the terminal input and output of text

Objectives (continued)

- Construct a simple Python program that performs inputs, calculations, and outputs
- Commenting - use docstrings to document Python programs
- Import functions from library modules – Math

Examples to show:

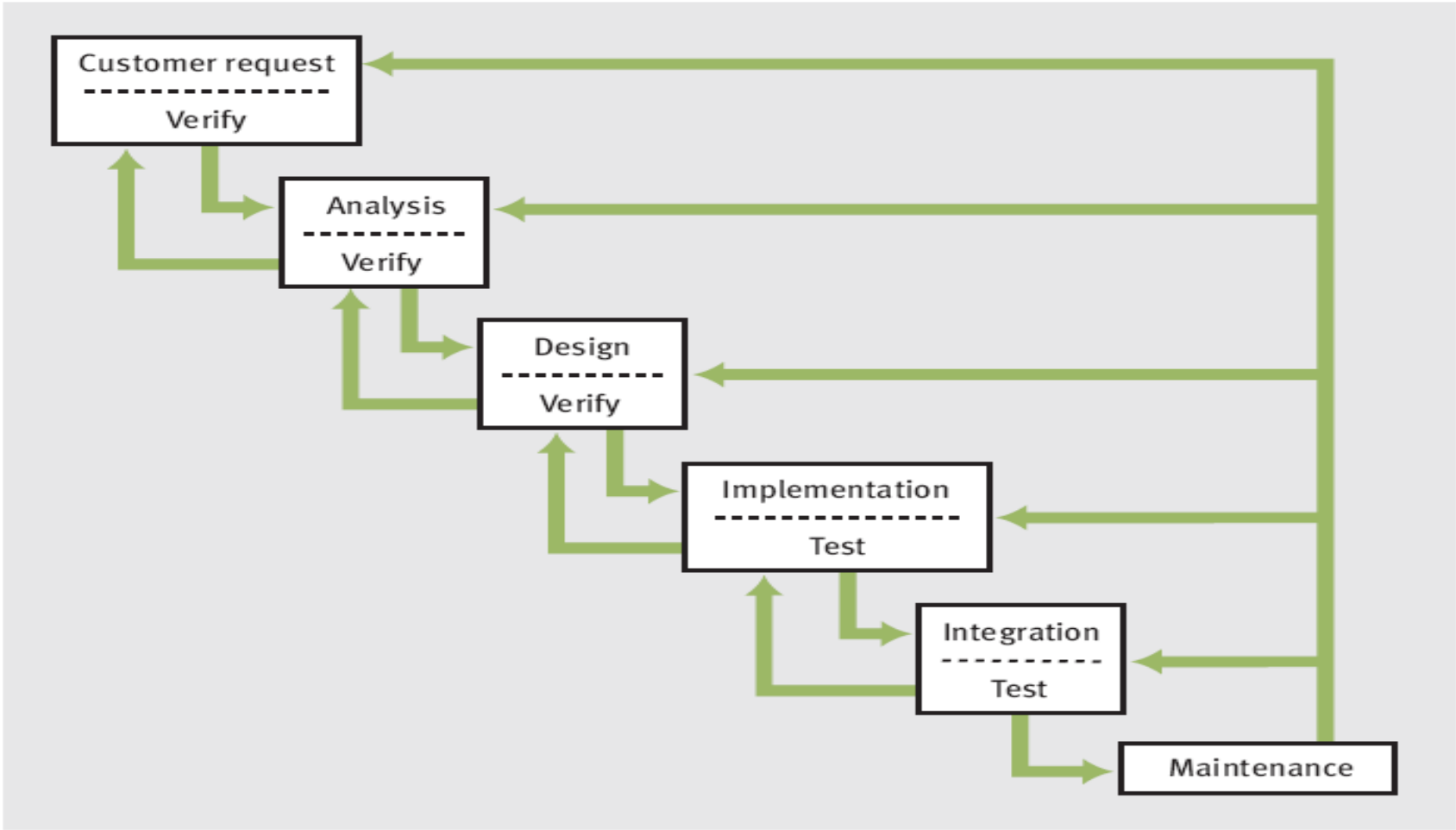
- How to write a program and save it with .py extension - (examples)
- Arithmetic calculations
- Check arithmetic precedence
- Show exponential reps
- Commenting
- Math function

The Software Development Process

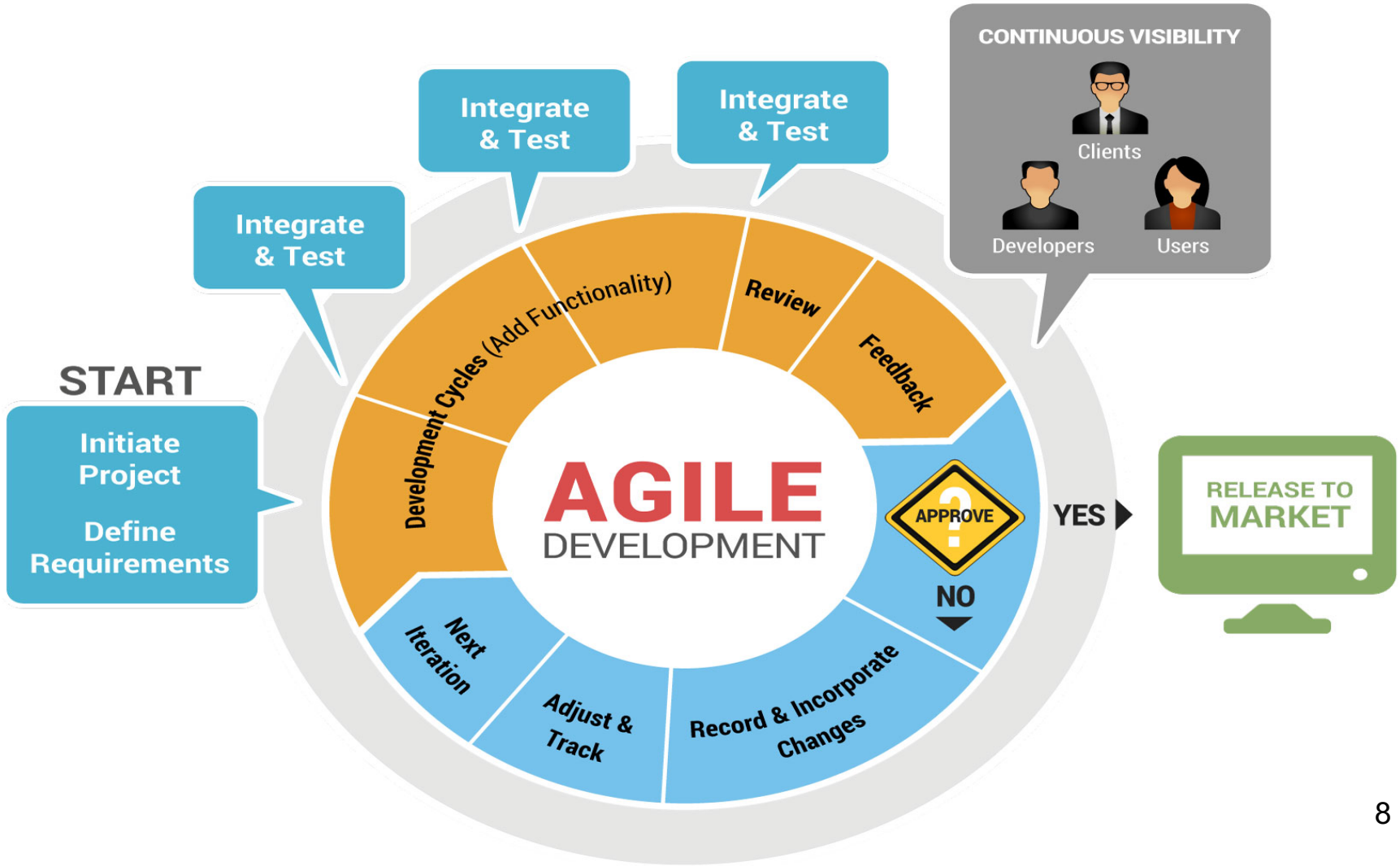
- **Software development:** process of planning and organizing a program
  - Several approaches; one is the **waterfall model**

- Another is - **incremental** and **iterative/Agile methodology**
  - Analysis and design may produce a **prototype** of a system for coding, and then back up to earlier phases to fill in more details after some testing

The Software Development Process (water fall)



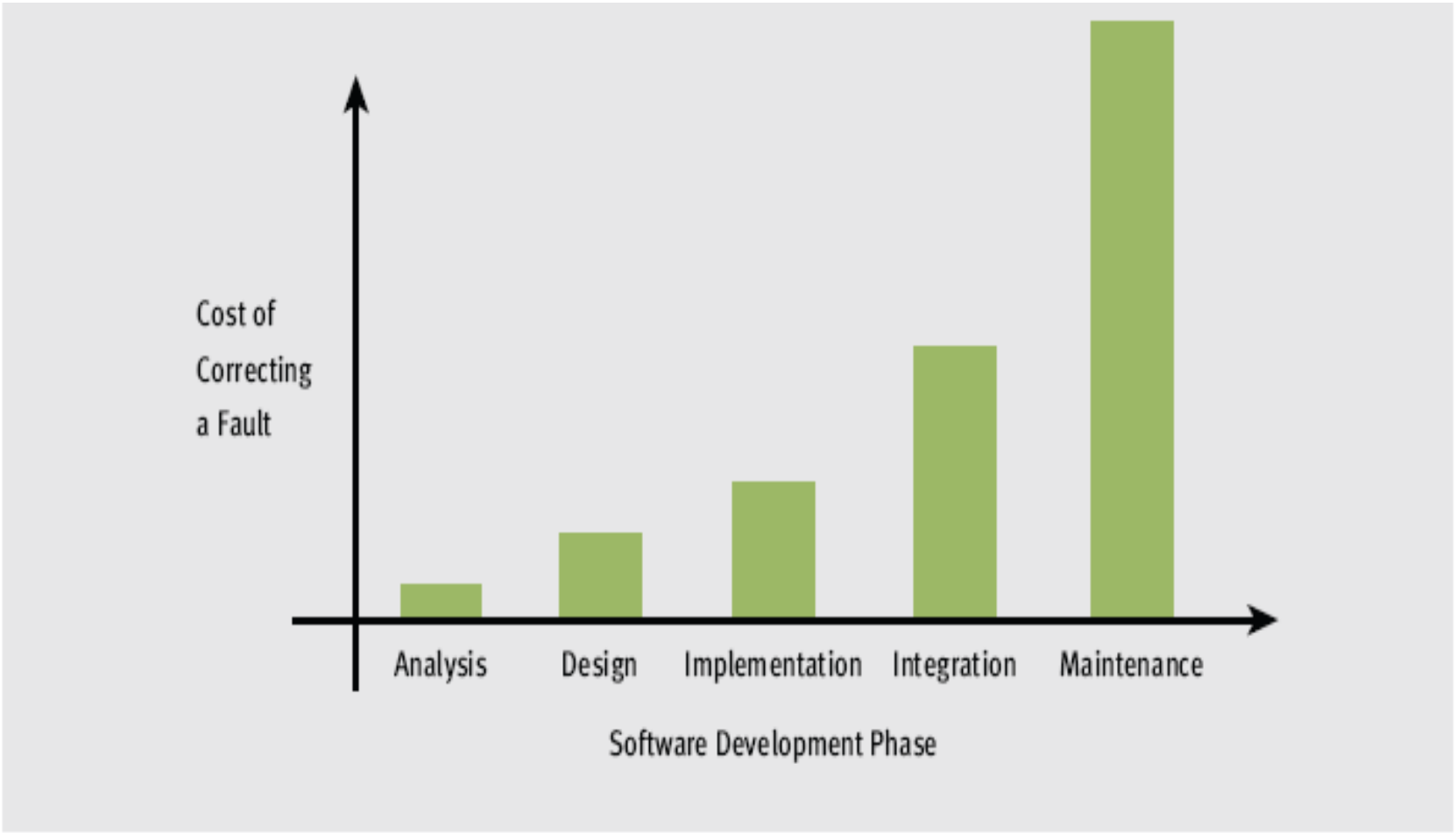
The Software Development Process (agile)



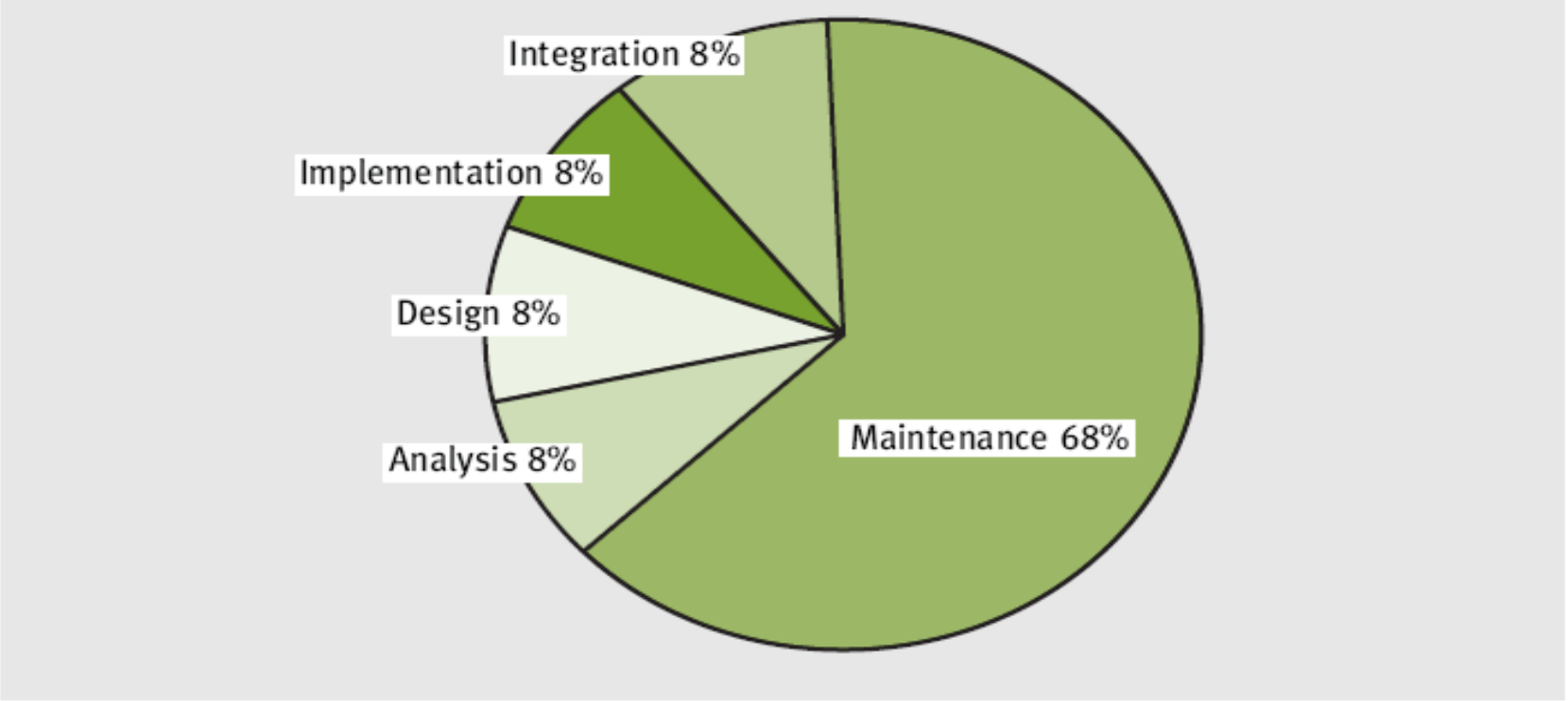
The Software Development Process (continued)

- Programs rarely work as hoped the first time they are run
  - Must perform extensive and careful testing
  - The cost of developing software is not spread equally over the phases

The Software Development Process (continued)



The Software Development Process (continued)



The Software Development Process (summary)

Notes:

1. Users usually don't know what they **need**. They **want** a lot of things. Make sure you gather requirements from stakeholders as accurately as possible
2. Analyse the Requirements – go back and forth
3. Make sure to prototype – get users to approve
4. Test – both unit testing and integrated testing !!!
5. Repeat all steps depending on the methodology

Data Types

- A **data type** consists of a set of values and a set of operations that can be performed on those values
- A **literal** is the way a value of a data type looks to a programmer e.g.; 4 5 0.5 “hi” etc….
- **int** and **float** are **numeric data types**

Data Types (continued)

TYPE OF DATA	PYTHON TYPE NAME	EXAMPLE LITERALS
Integers	`int`	`-1, 0, 1, 2`
Real numbers	`float`	`-0.55, .3333, 3.14, 6.0`
Character strings	`str`	`"Hi", "", 'A', '66'`

Variables and the Assignment Statement

- A **variable** associates a name with a value
  - Makes it easy to remember and use later in program
- Variable naming rules:
  - Reserved words cannot be used as variable names
    - Examples: **if**, **for**, **while** and **import**
  - Name must begin with a letter or underscore _
- The rest of the name can contain zero or more occurrences of the following things:
  - Digits (**0** to **9**) .
  - Alphabetic letters.
  - Underscores.

  - Names are case sensitive
    - Example: **WEIGHT** is different from **weight**
  - Tip: use "camel casing" (Example: **interestRate**)

Variables and Assignment Statement (continued)

- Nothing else is allowed in a variable name. Here is (incomplete) list of the things you can't use anywhere in a variable name:
  - Whitespace characters (the space, tab, new line).
  - Hyphens (-)
  - Quotation marks (single or double)
  - Symbol characters such as: the question mark, the exclamation mark, brackets, and so forth.
- The following words having special meaning in Python (reserve words), and so they cannot be used for variable

  Examples:

  def, if, for, while, else, except, return, True, range, list, continue

Variables and the Assignment Statement (continued)

- Programmers use all uppercase letters for **symbolic constants**
  - Examples: **TAX_RATE** and **STANDARD_DEDUCTION**

- Variables receive initial values and can be reset to new values with an **assignment statement**

  <variable name> = *<expression>*

  - Subsequent uses of the variable name in expressions are known as **variable references**

Variables : Quiz1

#Which of the following are valid variable names?

a. length
b. _width
c. firstWord
d. 2MoreToGo
e. halt!
f. tax Number

Solution: ????

Program Comments and Docstrings

- **What is a program comment?**

A comment is a piece of program text that the interpreter ignores but that provides useful documentation to programmers.

- **Why is it necessary?** A good programming style
  - Readability
  - Re-use

Docstring: It is a multi-line string that briefly describes parts of the program.

Program Comments and Docstrings

- **Docstring** example:

```
"""
Program: circle.py
Author: Ken Lambert
Last date modified: 2/10/11

The purpose of this program is to compute the area of a circle.
The input is an integer or floating-point number representing the
radius of the circle. The output is a floating-point number
labeled the area of the circle.
"""
```

- **End-of-line comment**: these comments begin with a # symbol and extend to the end of a line.
  - May explain the purpose of a variable or a strategy used in a piece of code.
- **End-of-line comment example**:

```
>>> rate = 4.5 # this variable provides a constant value
```

Program Comments and Docstrings

Good programming style:

1.Begin a program with a statement of its purpose and other information that helps the programmer or reader. This includes: what the program does, the authors name, version, date etc…

2.Accompany a variable definition with a comment that explains the variables purpose

3.Precede major segments of code with comments

4.Use comments to explain certain complex segments in your code

IDLE Editor

- Shell Command line
  - Does not allow changes

- Editor
  - Allows simplified coding
  - Save code (modules)
  - Run modules

Numeric Data Types and Character Sets

- The first applications of computers were to crunch numbers
- The use of numbers in many applications is still very important

- Numeric Data types integers, floating point numbers and their cousins Character sets

Integers

- Integers include 0, all of the positive whole numbers, all of the negative whole numbers
- Integer literals in Python are written without commas
- A leading negative sign indicates a negative value in python
- In real life, the range of **integers** is infinite
- A computer's memory places a limit on magnitude of the largest positive and negative integers - Python's **int** typical range: -2^{31} to $2^{31} - 1$ ie., (-2,147,483,648 to 2,147,483,647)
- A long integer is quite large. But still limited to your computers memory capacity.
  - Try evaluating: 2147483647 ** 100

Floating-Point Numbers

- A **real number** consists of a whole number, a decimal point and fractional part.
- Python uses **floating-point** numbers to represent real numbers.
- Python's **float** typical range: -10^{308} to 10^{308} and
- Typical precision: 16 digits
- A floating point number can be written using either ordinary decimal notation or **scientific notation**
- Scientific notation is usually useful when representing very large numbers

Floating-Point Numbers (continued)

DECIMAL NOTATION	SCIENTIFIC NOTATION	MEANING
3.78	3.78e0	3.78×10^{0}
37.8	3.78e1	3.78×10^{1}
3780.0	3.78e3	3.78×10^{3}
0.378	3.78e-1	3.78×10^{-1}
0.00378	3.78e-3	3.78×10^{-3}

Lets check this out !!!!

Data types: Quiz 2

#Write the values of the following floating point numbers in Python's scientific notation:

a.355.76

b.0.007832

c.4.3212

Data types: Quiz 2

#Write the values of the following floating point numbers in Python's scientific notation:

a.355.76

b.0.007832

c.4.3212

Solution:

a. 3.5576e2

b. 7.832e-3

c. 4.3212e0

Data types: Quiz 3

#Which data type would most appropriately be used to represent the following data values?

a. The number of months in a year
b. The area of a circle
c. The current minimum wage
d. The approximate age of the universe (12,000,000,000 yrs)
e. Your name

Data types : Quiz 3

#Which data type would most appropriately be used to represent the following data values?

a. The number of months in a year
b. The area of a circle
c. The current minimum wage
d. The approximate age of the universe (12,000,000,000 yrs)
e. Your name

Solution:

a. int
b. float
c. float
d. int
e. str

Character Sets

- Character literals in python look like strings and are of string types
- They belong to character sets – ASCII set (128 codes)
- ASCII set encodes each keyboard characters
  - American Standard Code for Information Interchange
  - The digits in the left column represent the leftmost digits of the ASCII Code.
  - The digit in the top row are the rightmost digits.
  - ASCII code for ‘R’ = 82

Character Sets

	0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DCI	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	SP	!	“	#	$	%	&	`
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	‘	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

Character Sets (continued)

- In Python, character literals look just like string literals and are of the string type
  - They belong to several different **character sets**, among them the **ASCII set**
  - ASCII character set maps to set of integers
- **ord** and **chr** convert characters to and from ASCII

```
>>> ord('a')
97
>>> ord('A')
65
>>> chr(65)
'A'
>>> chr(66)
'B'
>>>
```

Example: if you want to shift three places to the right of the letter 'A', simply write: chr(ord('A') + 3)

Expressions

- Variable reference evaluates to the variable's current value
- **Expressions** provide easy way to perform operations on data values to produce other values
- When entered at Python shell prompt:
  - an expression's operands are evaluated
  - its operator is then applied to these values to compute the value of the expression

Arithmetic Expressions

- An **arithmetic expression** consists of operands and operators combined in a manner that is already familiar to you from learning algebra

OPERATOR	MEANING	SYNTAX
-	Negation	`-a`
**	Exponentiation	`a ** b`
*	Multiplication	`a * b`
/	Division	`a / b`
//	Quotient	`a // b`
%	Remainder or modulus	`a % b`
+	Addition	`a + b`
-	Subtraction	`a - b`

Arithmetic Expressions (continued)

- **Precedence rules**:
  - ** has the highest precedence and is evaluated first
  - Unary negation is evaluated next
  - *, /, and % are evaluated before + and -
  - + and - are evaluated before equal to (=)
  - With two exceptions, operations of equal precedence are **left associative**, so they are evaluated from left to right
    - Exponentiation (**) and assignment (=) are **right associative**
  - You can use parenthesis ( ) to change the order of evaluation as parenthesis takes precedence.

Arithmetic Expressions (continued)

EXPRESSION	EVALUATION	VALUE
5 + 3 * 2	5 + 6	11
(5 + 3) * 2	8 * 2	16
6 % 2	0	0
2 * 3 ** 2	2 * 9	18
-3 ** 2	-(3 ** 2)	-9
-(3) ** 2	9	9
2 ** 3 ** 2	2 ** 9	512
(2 ** 3) ** 2	8 ** 2	64
45 / 0	Error: cannot divide by 0	
45 % 0	Error: cannot divide by 0	

Syntax error: set of rules for constructing well formed expressions in a language (error when an expression or sentence is not well formed).

Semantic error: detected when the action that an expression describes cannot be carried out, even if the expression is syntactically correct.

Example: 45%0 is a **semantic error**

Arithmetic calculations – Quiz 4

#Let x = 8 and y = 2. Write the values of the following expressions:

a. x + y * 3
b. (x + y) * 3
c. x ** y
d. x % y
e. x / 12.0
f. x // 6

Arithmetic calculations – Quiz 4:

#Let x = 8 and y = 2. Write the values of the following expressions:

a. x + y * 3 #14

b. (x + y) * 3 #30

c. x ** y #64

d. x % y #0

e. x / 12.0 #0.6666666666666663

f. x // 6 #1

Variables and the Assignment Statement – question !

#It is typical that you have a first name and a surname. Write a python program that prints the first name followed by the surname, making sure that there is a blank space between the first name and the surname. Your program should store the first name in a variable called firstName and the surname in a variable secondName. You may use additional variables for the task.

Arithmetic Expressions (continued)

- When **both operands of an expression are of the same numeric type**, the resulting value is also of that type
- When each operand is of a different type, the resulting value is of the more general type
  - Example: **3 / 4** is **0**, whereas **3 / 4.0** is **.75**

```
>>> 3 + 4 * \
2 ** 5
131
>>>
```

Note: For multi-line expressions, use a \

Mixed-Mode Arithmetic and Type Conversions

- **Mixed-mode arithmetic** involves integers and floating-point numbers:

```
>>> 3.14 * 3 ** 2
28.26
```

- **Remember**—Python has different operators for quotient and exact division:

`3 // 2 * 5.0` yields `1 * 5.0`, which yields `5.0`

`3 / 2 * 5` yields `1.5 * 5`, which yields `7.5`

Tip:

  - Use exact division
  - Use a **type conversion function** with variables

Using Functions and Modules - intro

- Python includes many useful functions, which are organized in libraries of code called **modules**

Calling Functions - Intro: Arguments and Return Values

- A **function** is chunk of code that can be called by name to perform a task
- Functions often require **arguments** or **parameters**
  - Arguments may be **optional** or **required**
- When function completes its task, it may **return a value** back to the part of the program that called it

```
>>> help(round)

Help on built-in function round in module builtin:

round(...)
    round(number[, ndigits]) -> floating point number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the same type as
    number. ndigits may be negative.
```

The math Module

```
>>> import math
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atanh', 'ceil','copysign', 'cos', 'cosh', 'degrees', 'e',
'exp', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'hypot',
'isinf', 'isnan', 'ldexp', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

- To use a resource from a module, you write the name of a module as a qualifier, followed by a dot (.) and the name of the resource
  - Example: **math.pi**

```
>>> math.pi
3.1415926535897931
>>> math.sqrt(2)
1.4142135623730951
```

The math Module (continued)

- You can avoid the use of the qualifier with each reference by importing the individual resources

```
>>> from math import pi, sqrt
>>> print(pi, sqrt(2))
3.14159265359 1.41421356237
>>>
```

- You may import all of a module's resources to use without the qualifier
  - Example: **`from math import *`**

Mixed-Mode Arithmetic and Type Conversions (continued)

CONVERSION FUNCTION	EXAMPLE USE	VALUE RETURNED
int(<a number or a string>)	int(3.77)	3
	int("33")	33
float(<a number or a string>)	float(22)	22.0
str(<any value>)	str(99)	'99'

Mixed-Mode Arithmetic and Type Conversions (continued)

- Note that the **int** function converts a **float** to an **int** by truncation, not by rounding

```
>>> int(6.75)
6
>>> round(6.75)
7
```

String Literals

- A string literal is a sequence of characters enclosed in single or double quotation marks
- '' and "" represent the **empty string**
- Use ''' and """ for multi-line paragraphs

```
>>> "I'm using a single quote in this string!"
"I'm using a single quote in this string!"
>>> print("I'm using a single quote in this string!")
I'm using a single quote in this string!
>>>
>>> print("""This very long sentence extends all the way to
the next line.""")
This very long sentence extends all the way to
the next line.
```

```
>>> """This very long sentence extends all the way to
the next line. """
'This very long sentence extends all the way to\nthe next line.'
>>>
```

Escape Sequences

- The newline character **\n** is called an **escape sequence**

ESCAPE SEQUENCE	MEANING
\b	Backspace
\n	Newline
\t	Horizontal tab
\\	The \ character
\'	Single quotation mark
\"	Double quotation mark

String Concatenation

- You can join two or more strings to form a new string using the concatenation operator **+**
- The * operator allows you to build a string by repeating another string a given number of times

```
>>> " " * 10 + "Python"
'          Python'
>>>
```

Mixed-Mode Arithmetic and Type Conversions (continued)

- Type conversion also occurs in the construction of strings from numbers and other strings

```
>>> profit = 1000.55
>>> print('$' + profit)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'float' objects
```

- Solution: use **`str`** function

```
>>> print('$' + str(profit))
$1000.55
```

- Python is a strongly **typed** programming language

Summary

- Waterfall model describes software development process in terms of several phases
- Literals are data values that can appear in program
- The string data type is used to represent text for input and output
- Escape characters begin with backslash and represent special characters such as delete key
- A docstring is string enclosed by triple quotation marks and provides program documentation

Summary (continued)

- Comments are pieces of code not evaluated by the interpreter but can be read by programmers to obtain information about a program
- Variables are names that refer to values
- Some data types: **int** and **float**
- Arithmetic operators are used to form arithmetic expressions
  - Operators are ranked in precedence
- Mixed-mode operations involve operands of different numeric data types

Summary (continued)

- A function call consists of a function's name and its arguments or parameters
  - May return a result value to the caller
- Python is a strongly typed language
- A module is a set of resources
  - Can be imported
- A semantic error occurs when the computer cannot perform the requested operation
- A logic error produces incorrect results